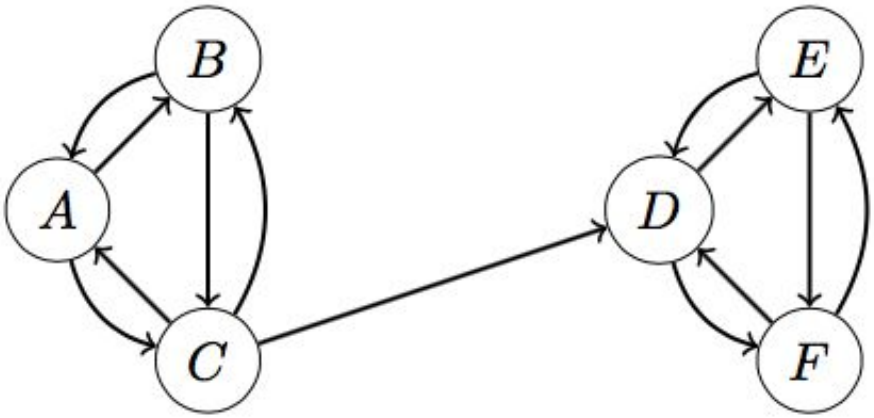
Thought experiment

- Run DFS starting at D:



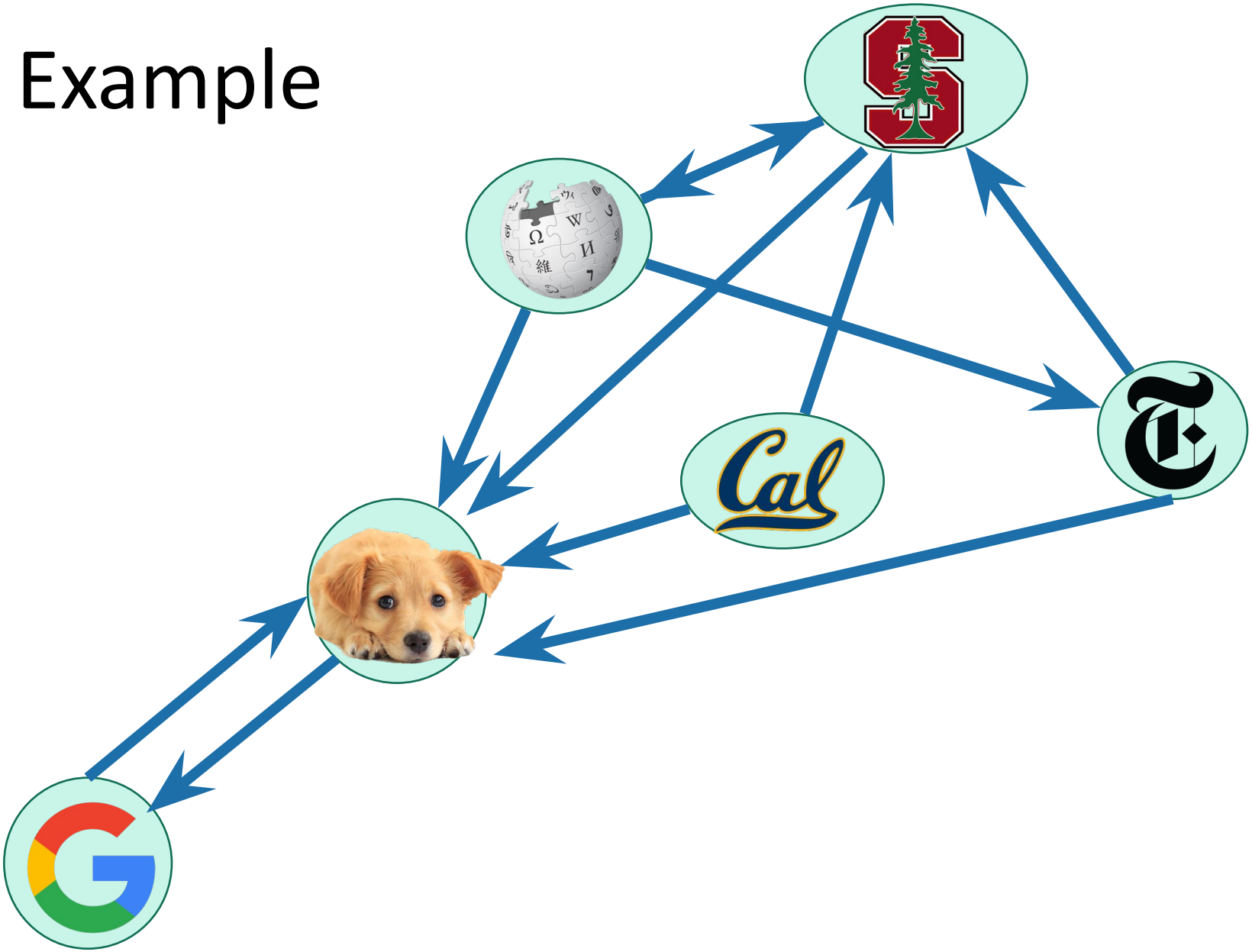
- That will identify SCCs…
- Issues:
  - How do we know where to start DFS?
  - It wouldn't have found the SCCs if we started from A.

Algorithm

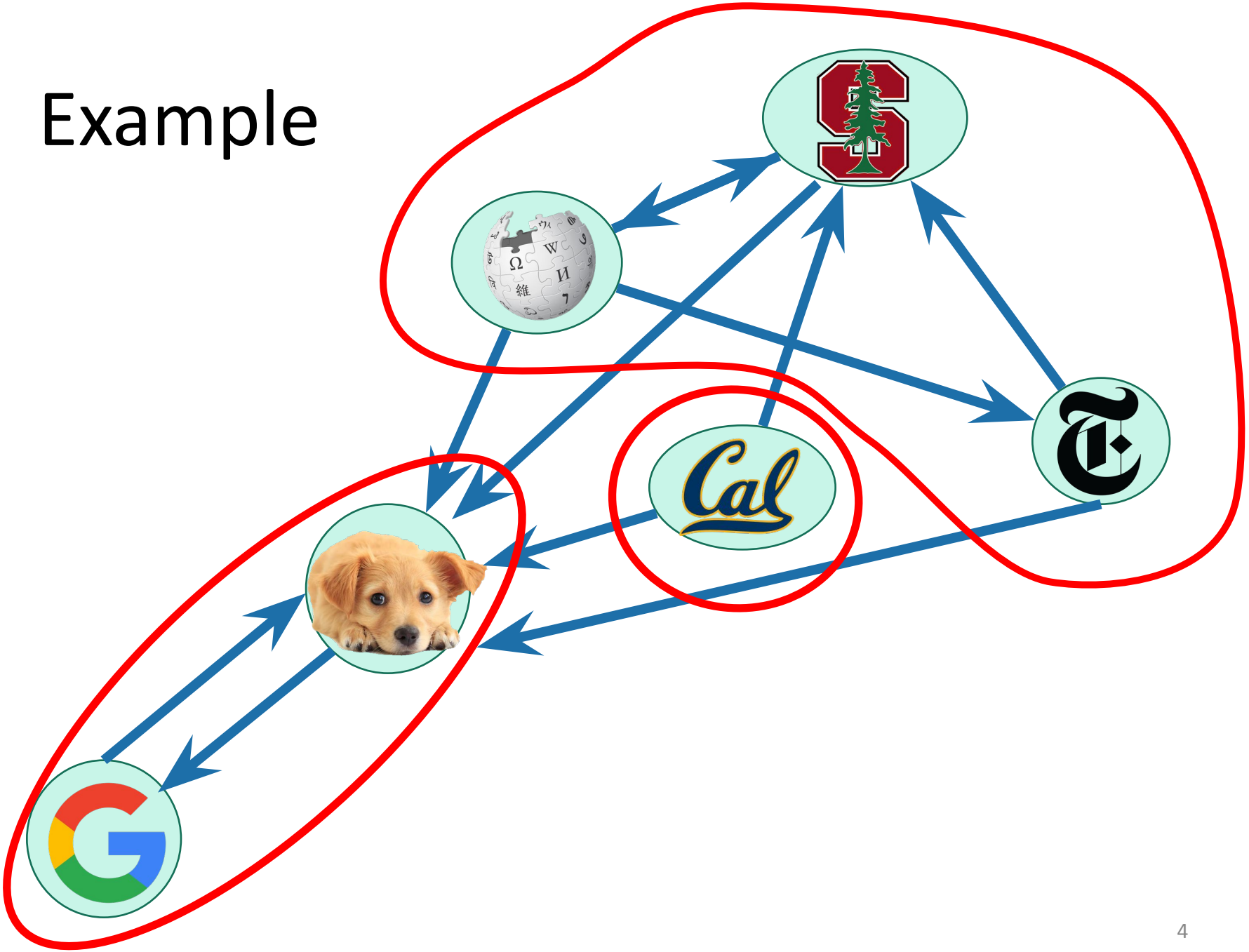
Running time: O(n + m)

- Do DFS to create a DFS forest.
  - Choose starting vertices in any order.
  - Keep track of finishing times.
- Reverse all the edges in the graph.
- Do DFS again to create **another DFS forest**.
  - This time, order the nodes in the reverse order of the finishing times that they had from the first DFS run.
- The SCCs are the different trees in the **second DFS forest.**

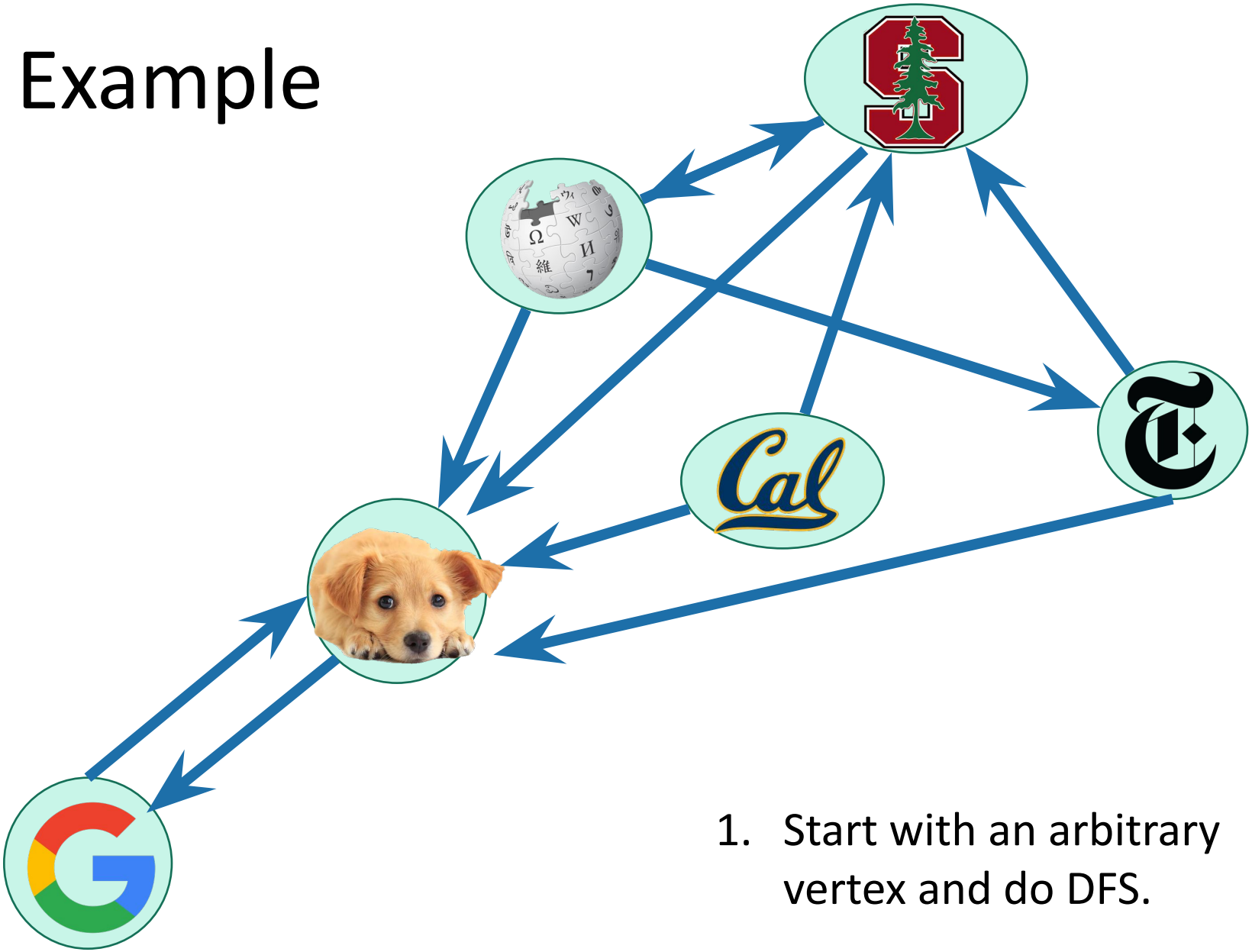
Example



Example

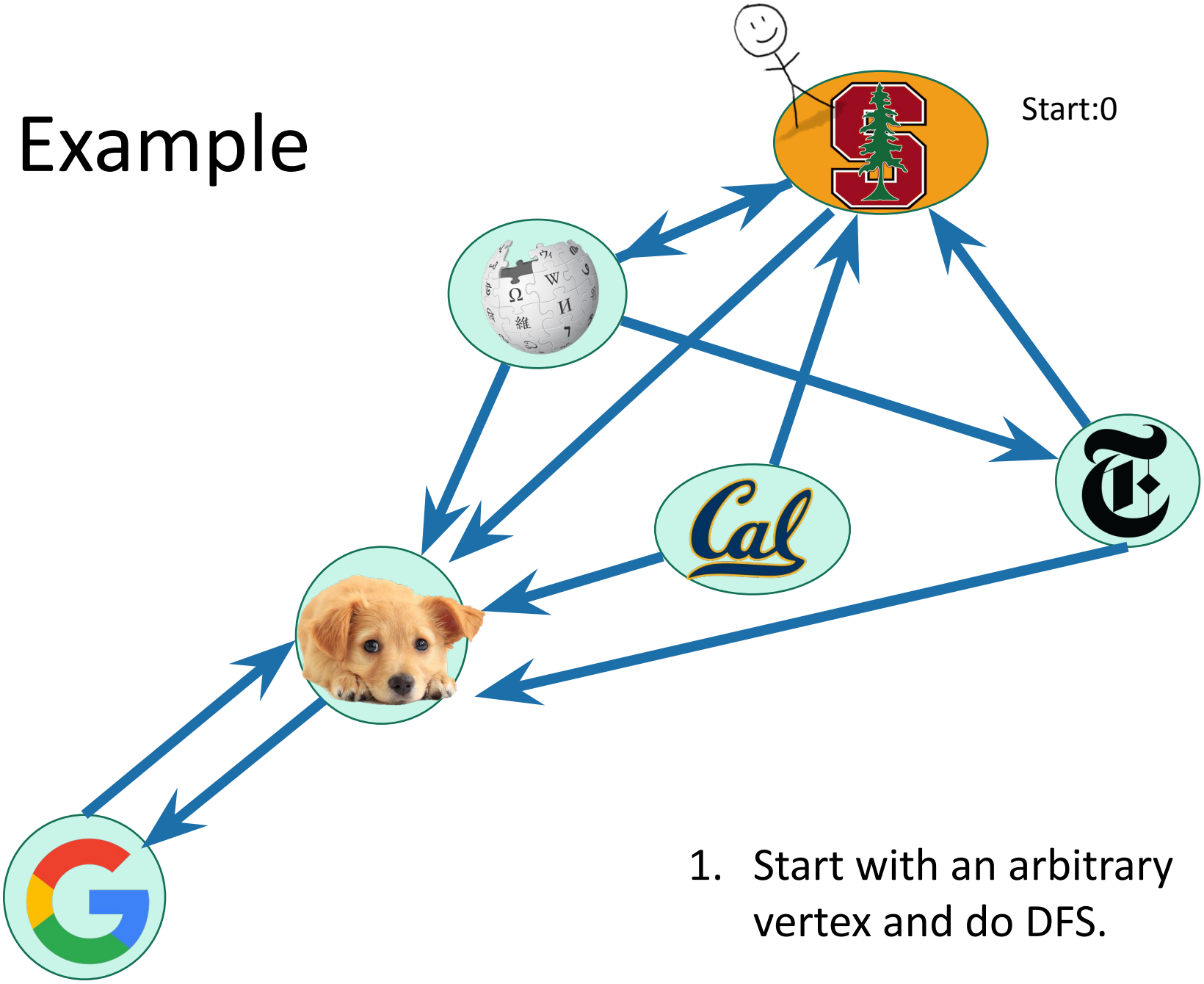


Example



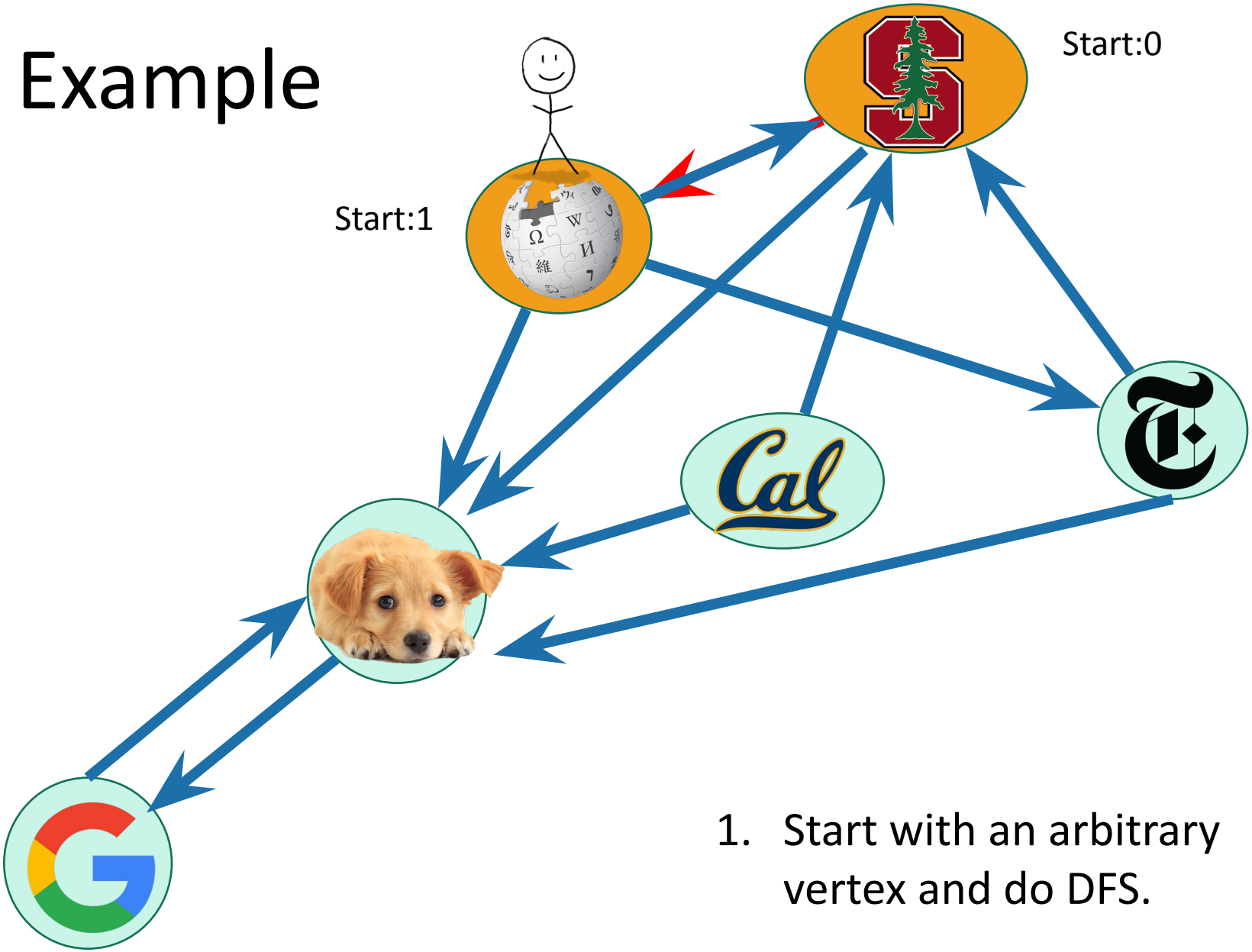
1. Start with an arbitrary vertex and do DFS.

Example



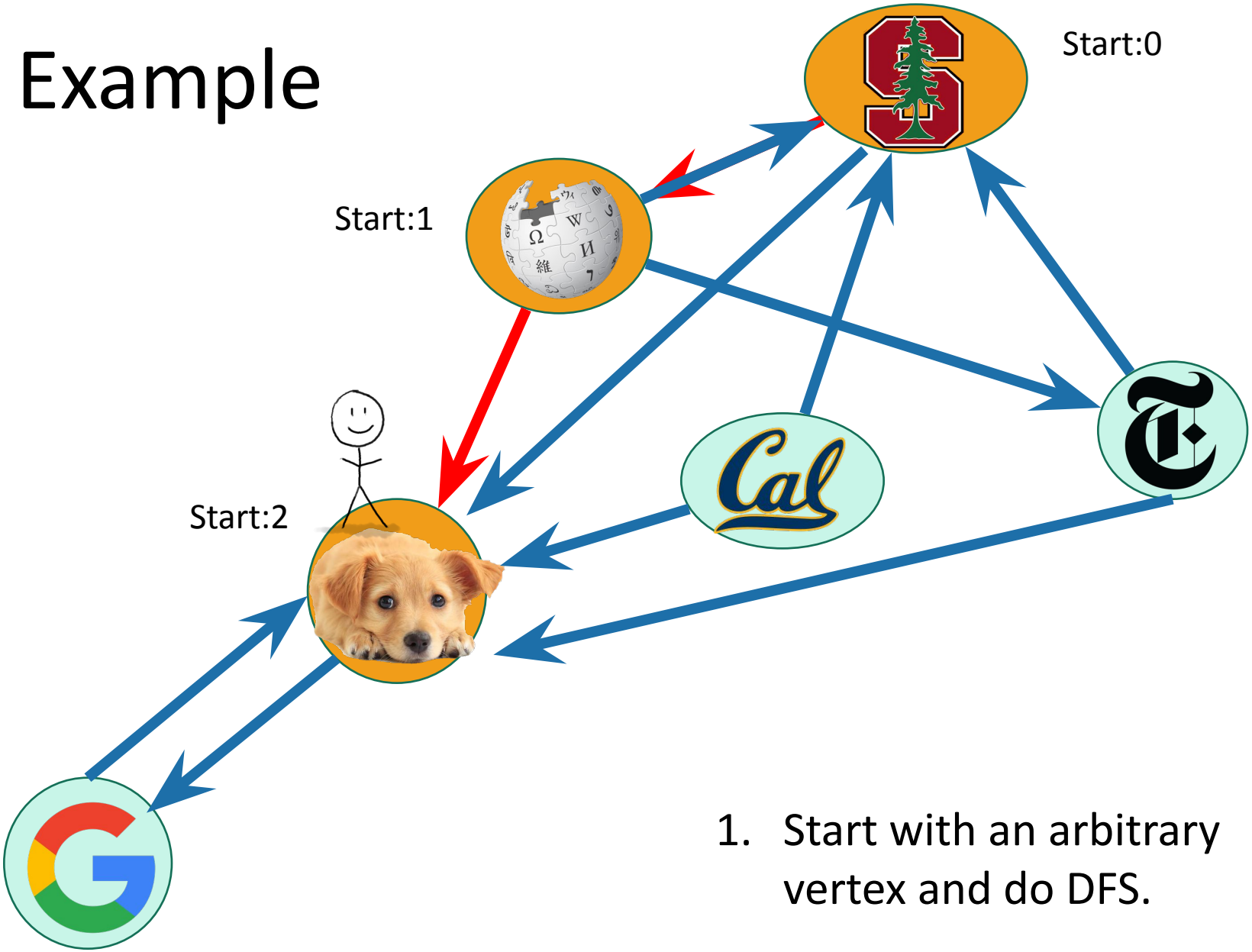
1. Start with an arbitrary vertex and do DFS.

Example



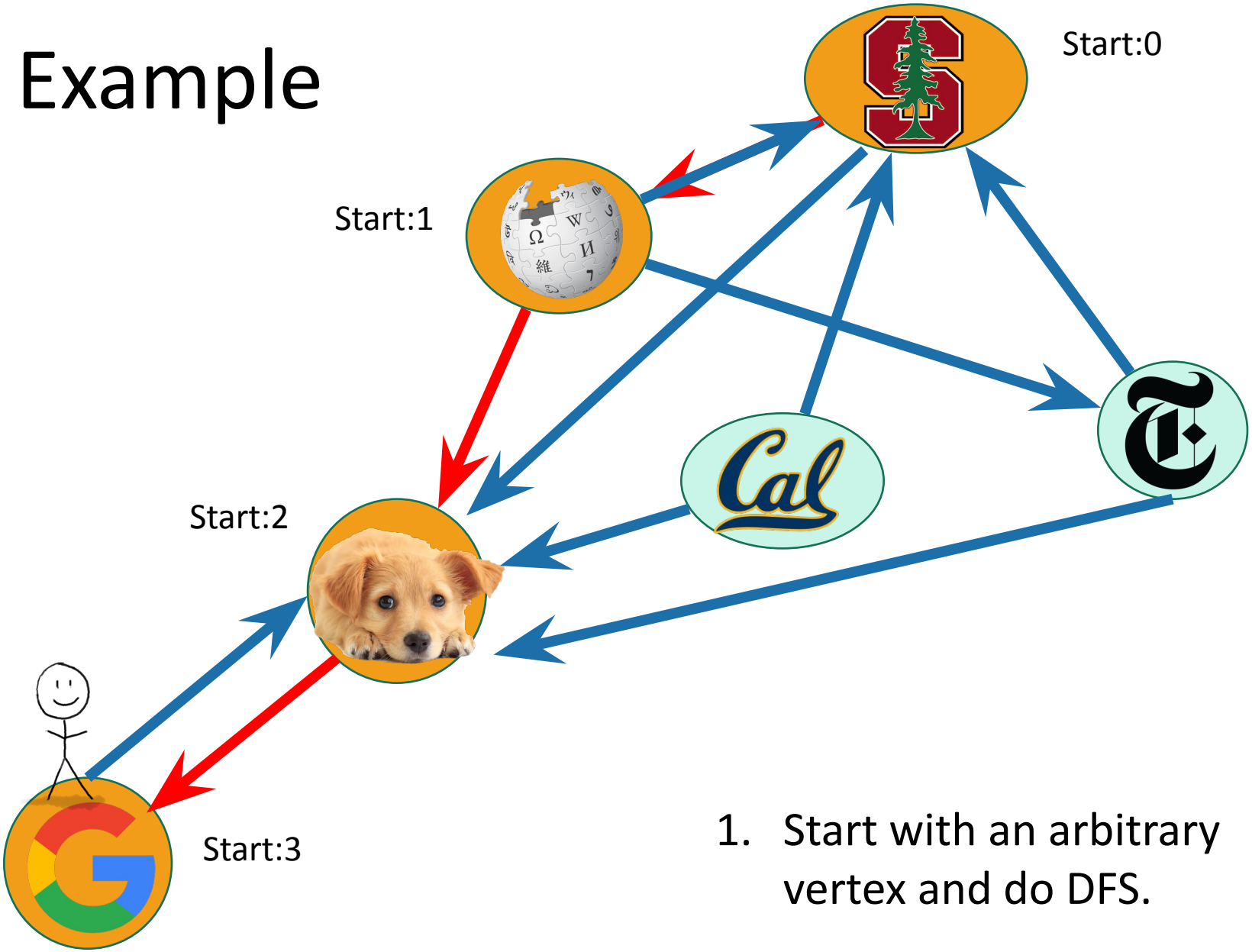
1. Start with an arbitrary vertex and do DFS.

Example



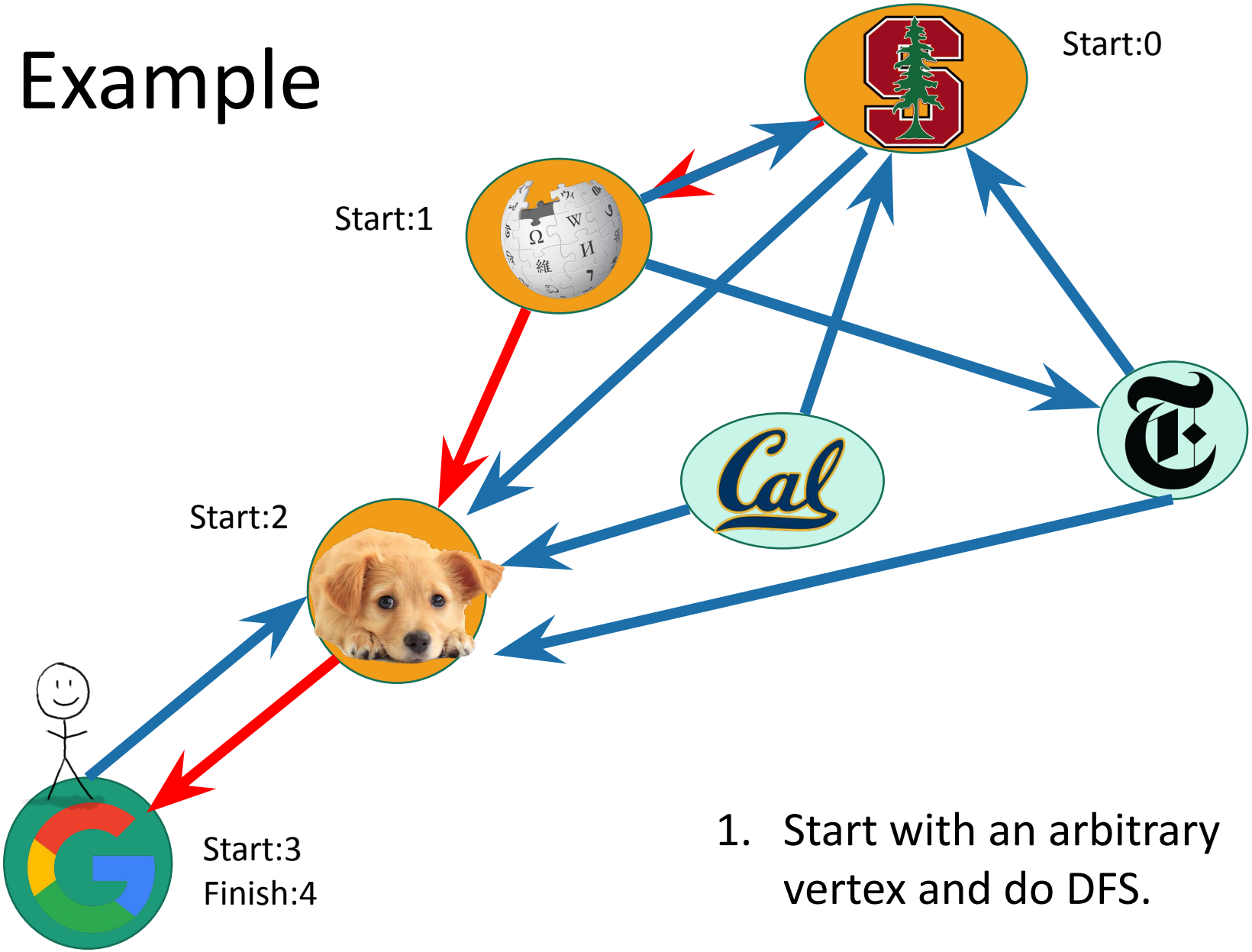
1. Start with an arbitrary vertex and do DFS.

Example



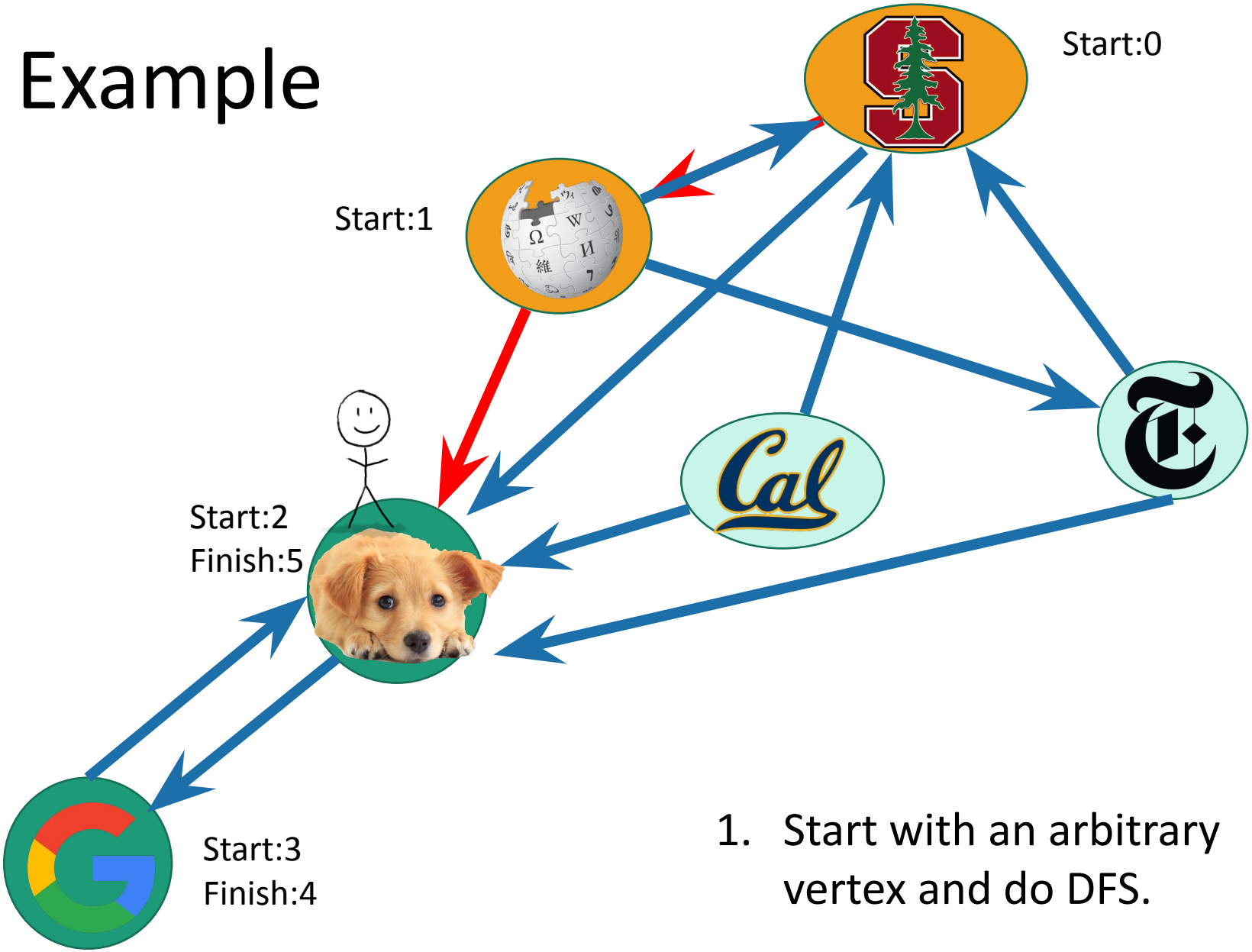
1. Start with an arbitrary vertex and do DFS.

Example



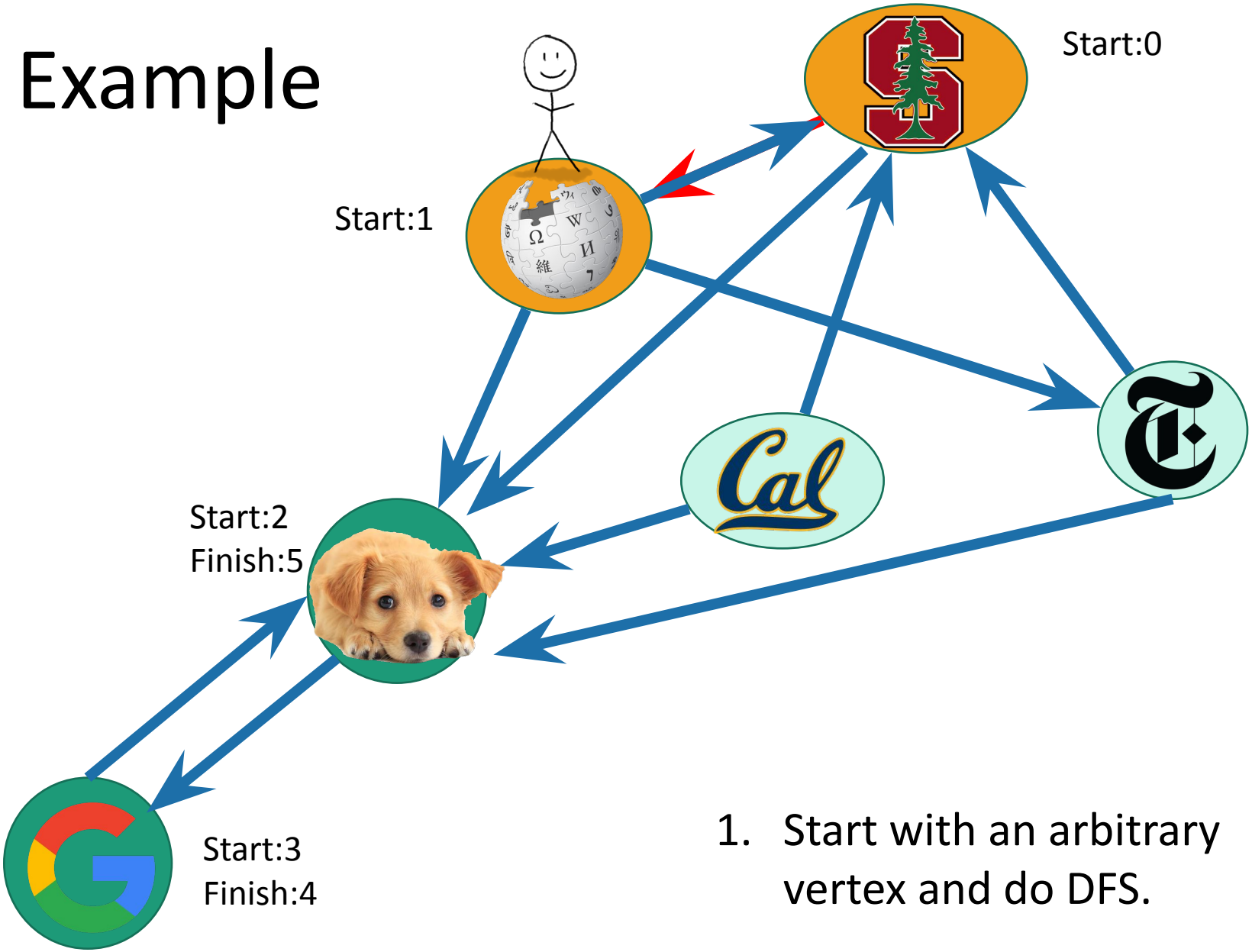
1. Start with an arbitrary vertex and do DFS.

Example



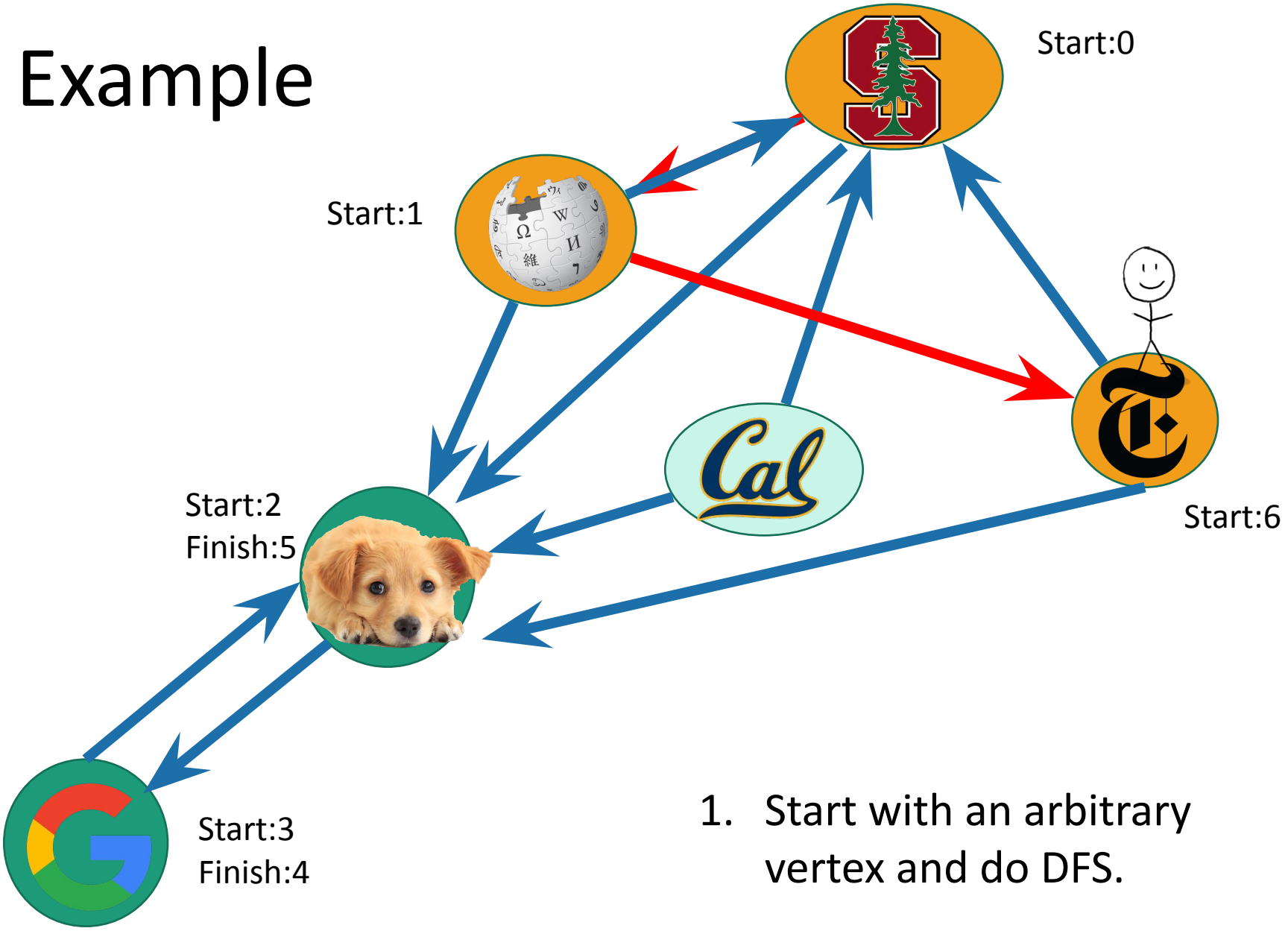
1. Start with an arbitrary vertex and do DFS.

Example



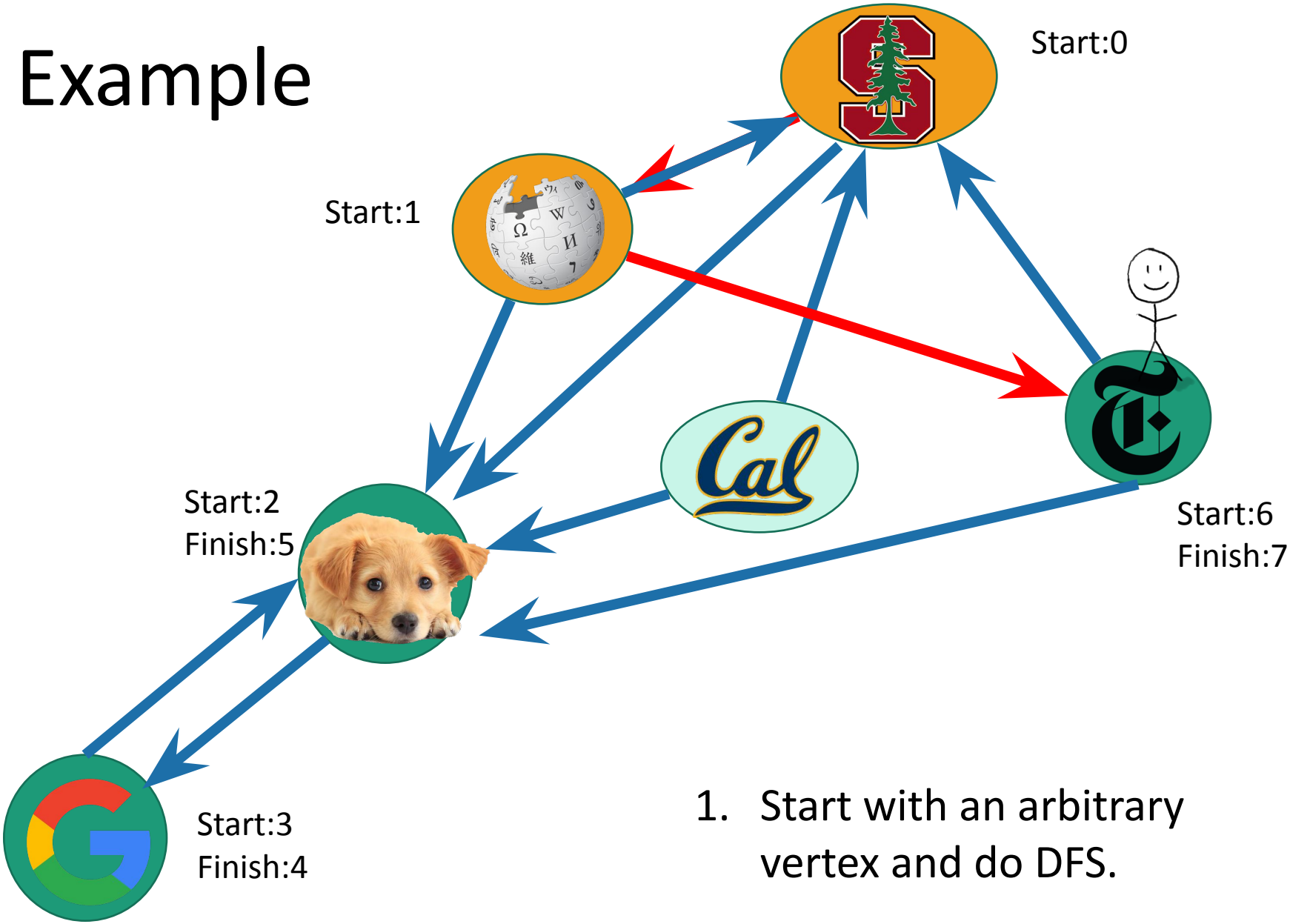
1. Start with an arbitrary vertex and do DFS.

Example



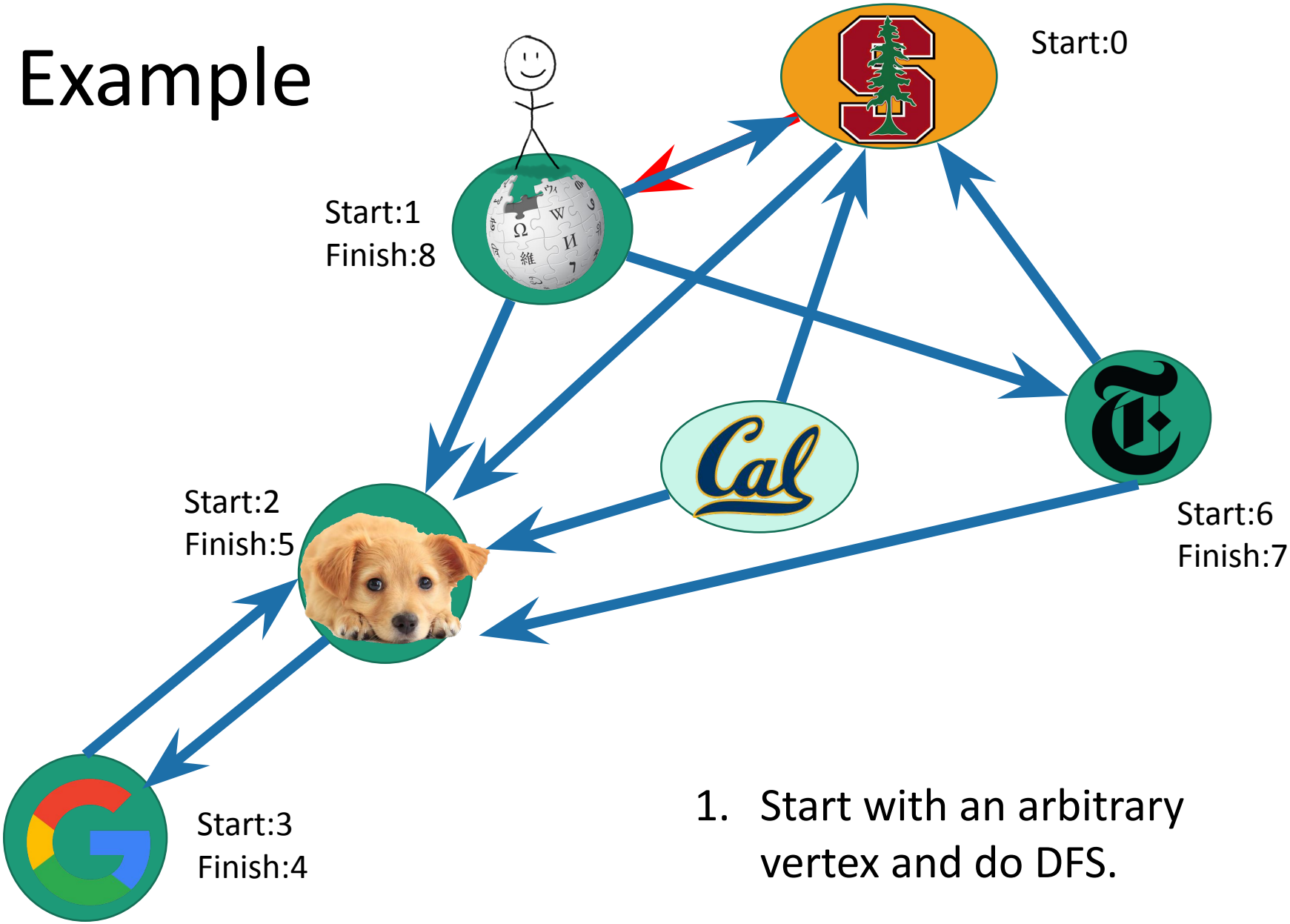
1. Start with an arbitrary vertex and do DFS.

Example



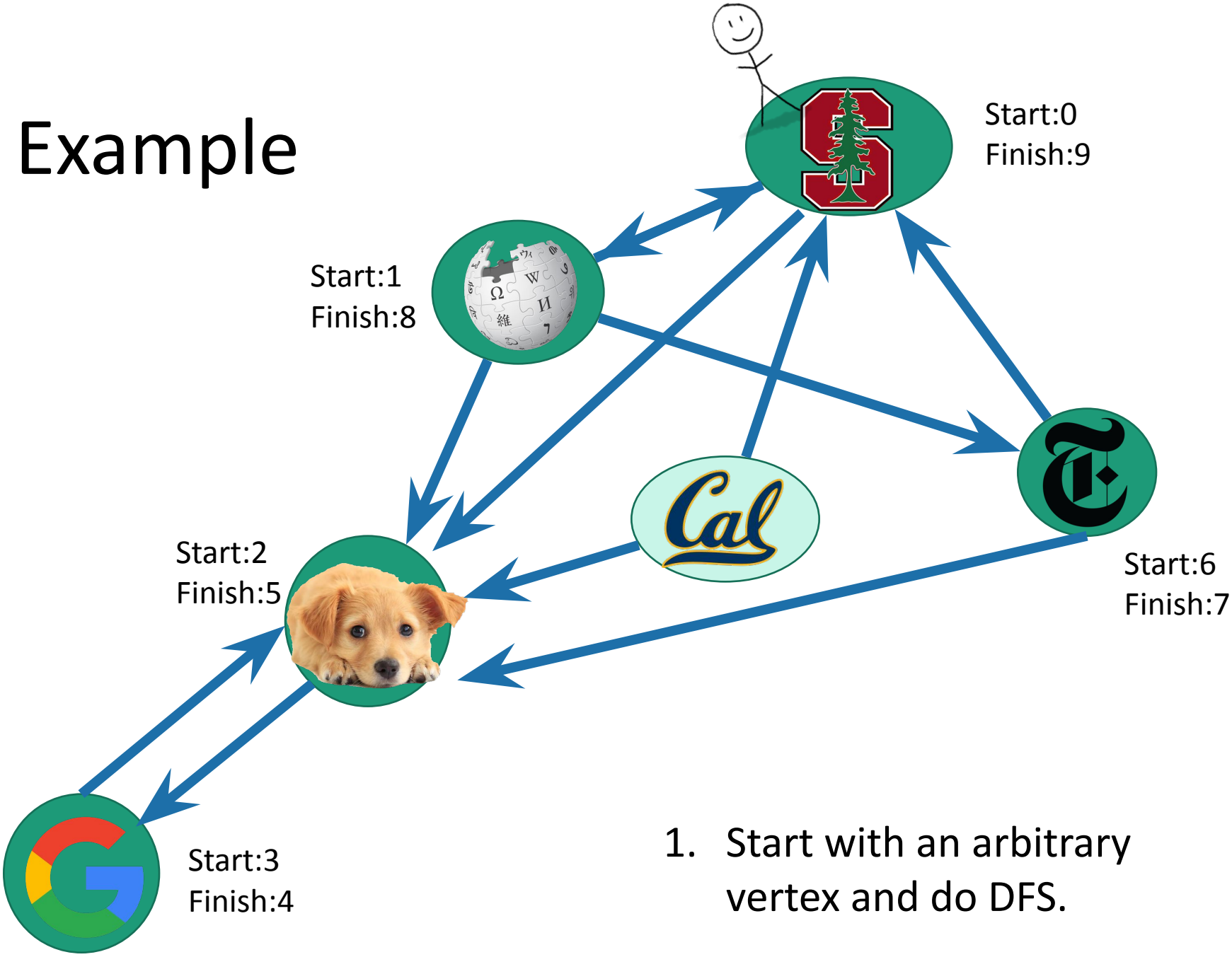
1. Start with an arbitrary vertex and do DFS.

Example



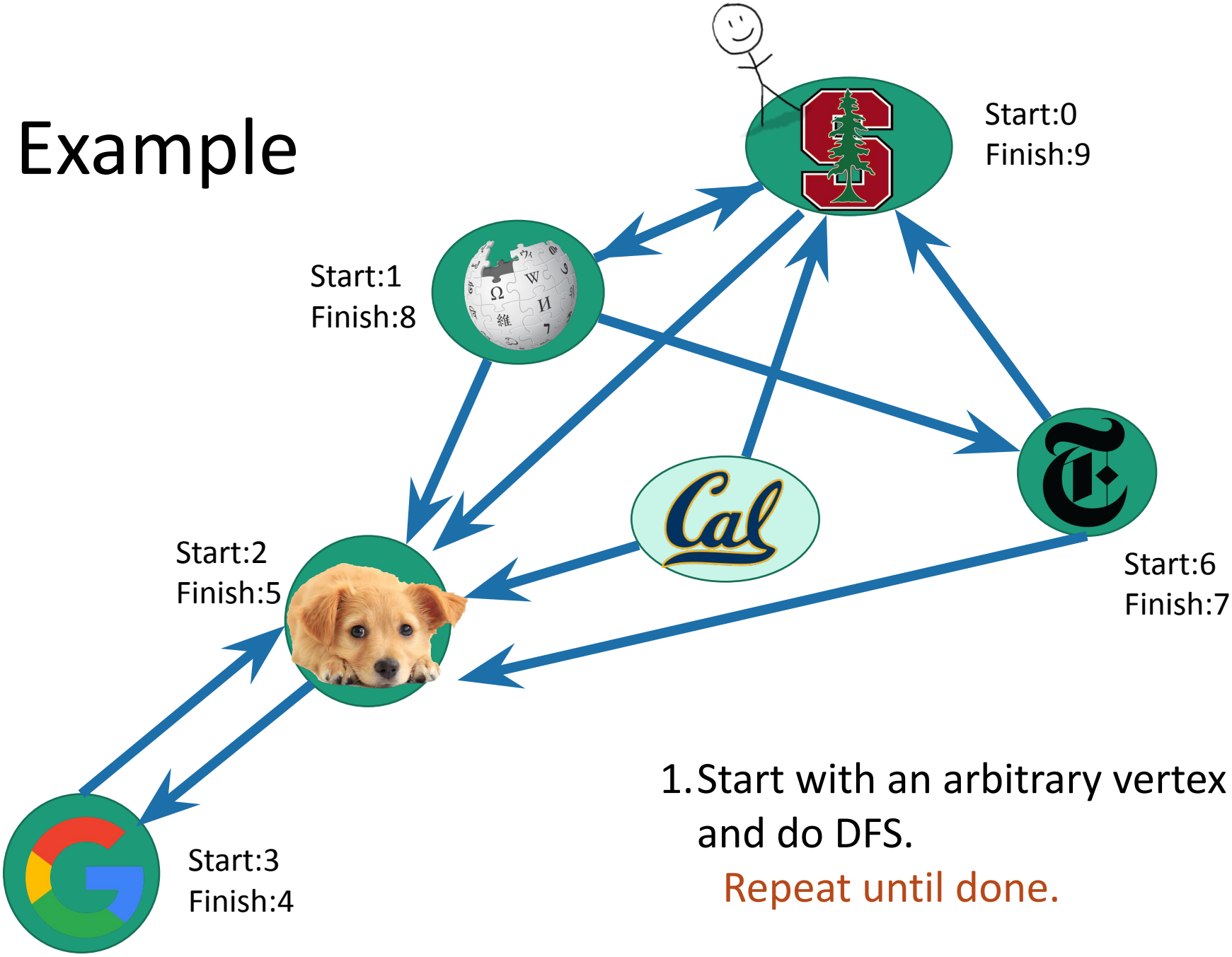
1. Start with an arbitrary vertex and do DFS.

Example



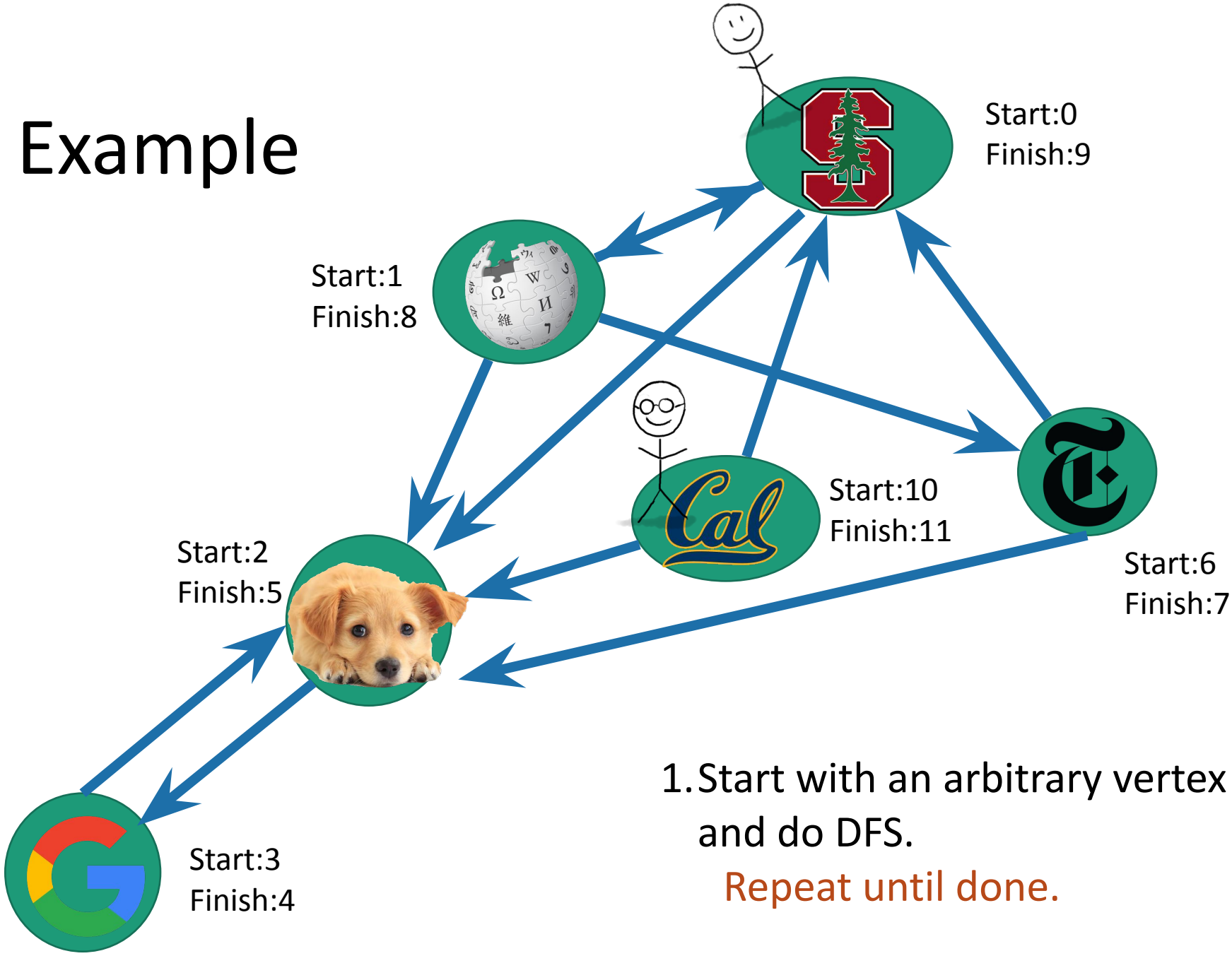
1. Start with an arbitrary vertex and do DFS.

Example



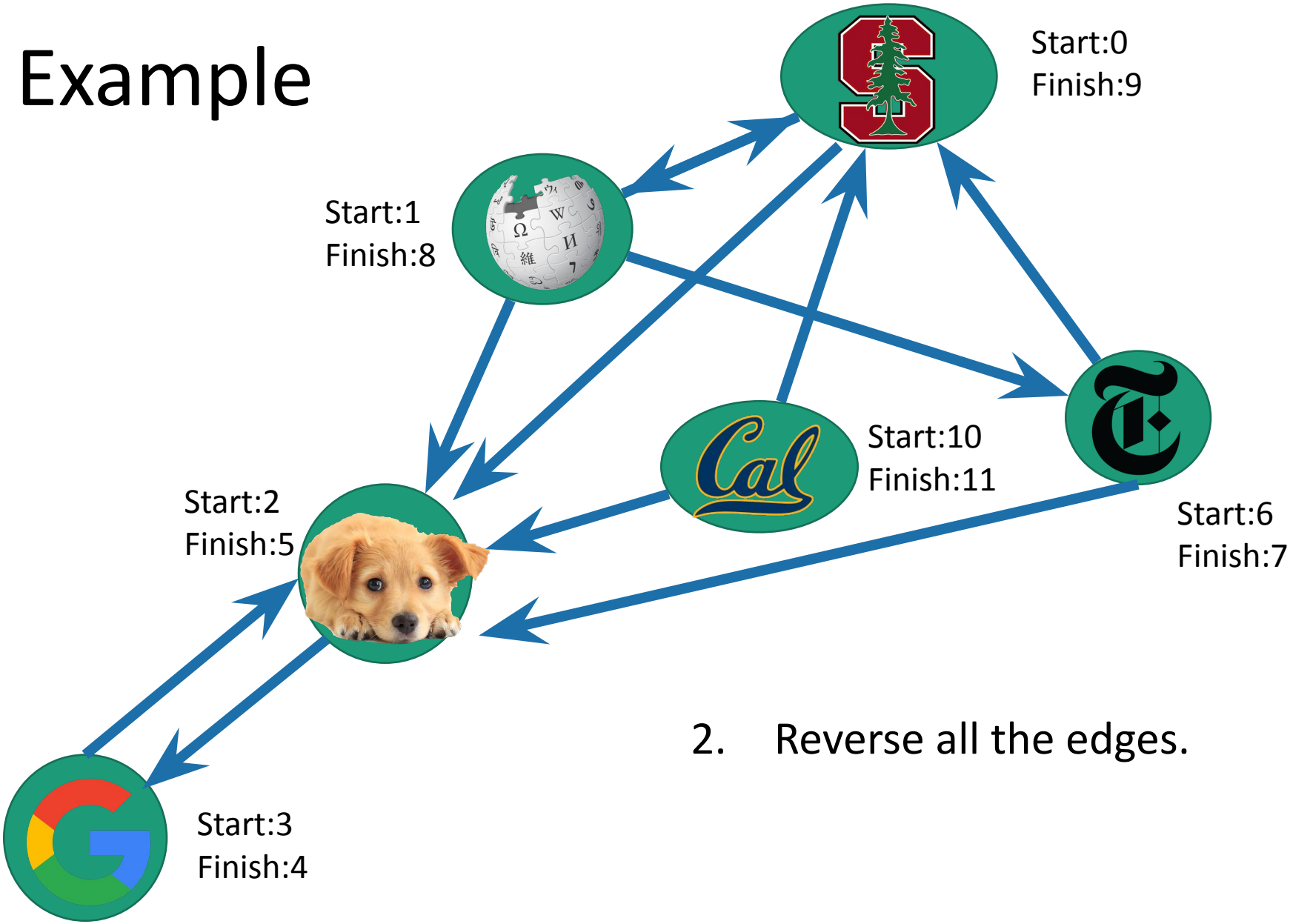
1. Start with an arbitrary vertex and do DFS.
   Repeat until done.

Example



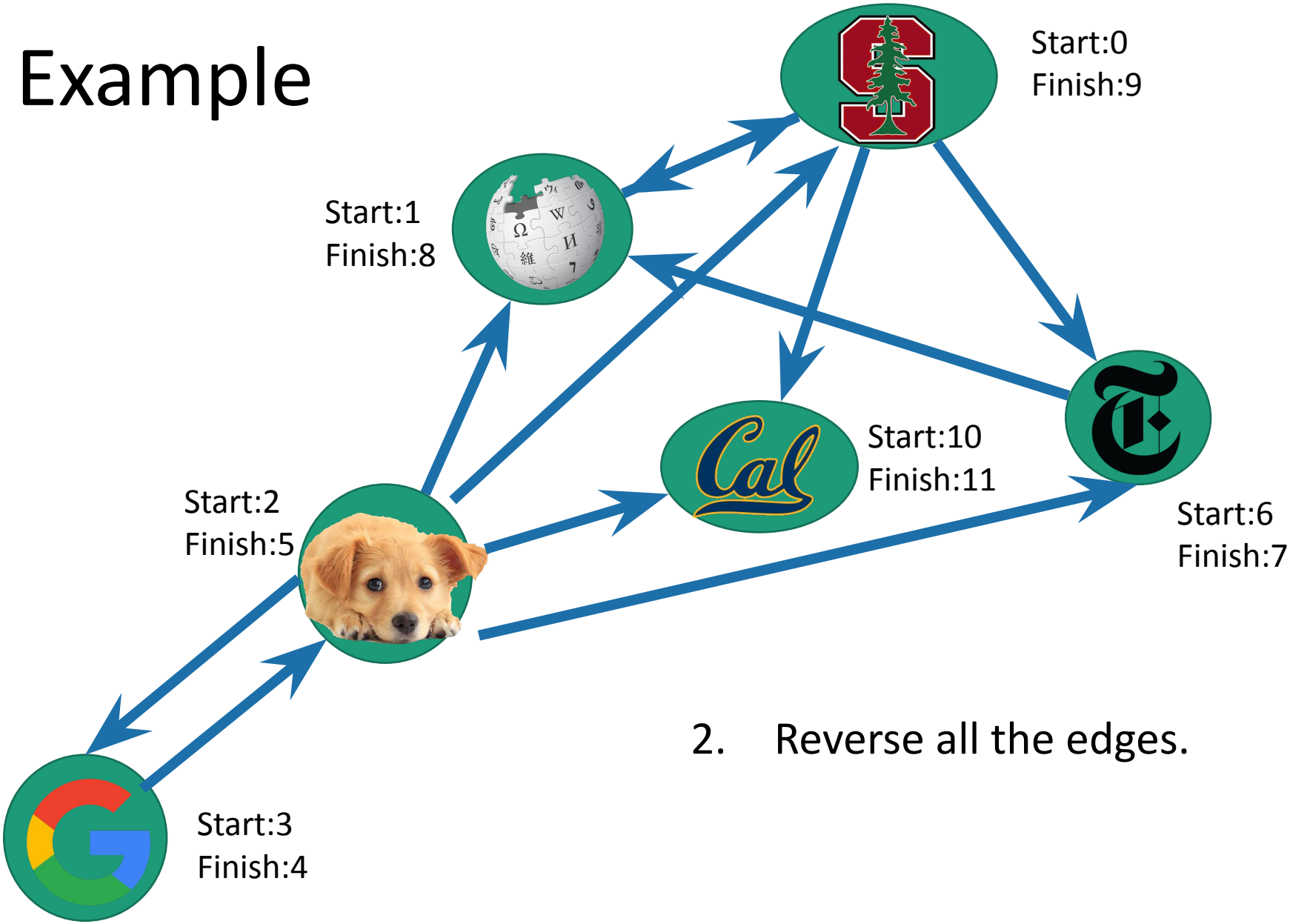
1. Start with an arbitrary vertex and do DFS.
   Repeat until done.

Example



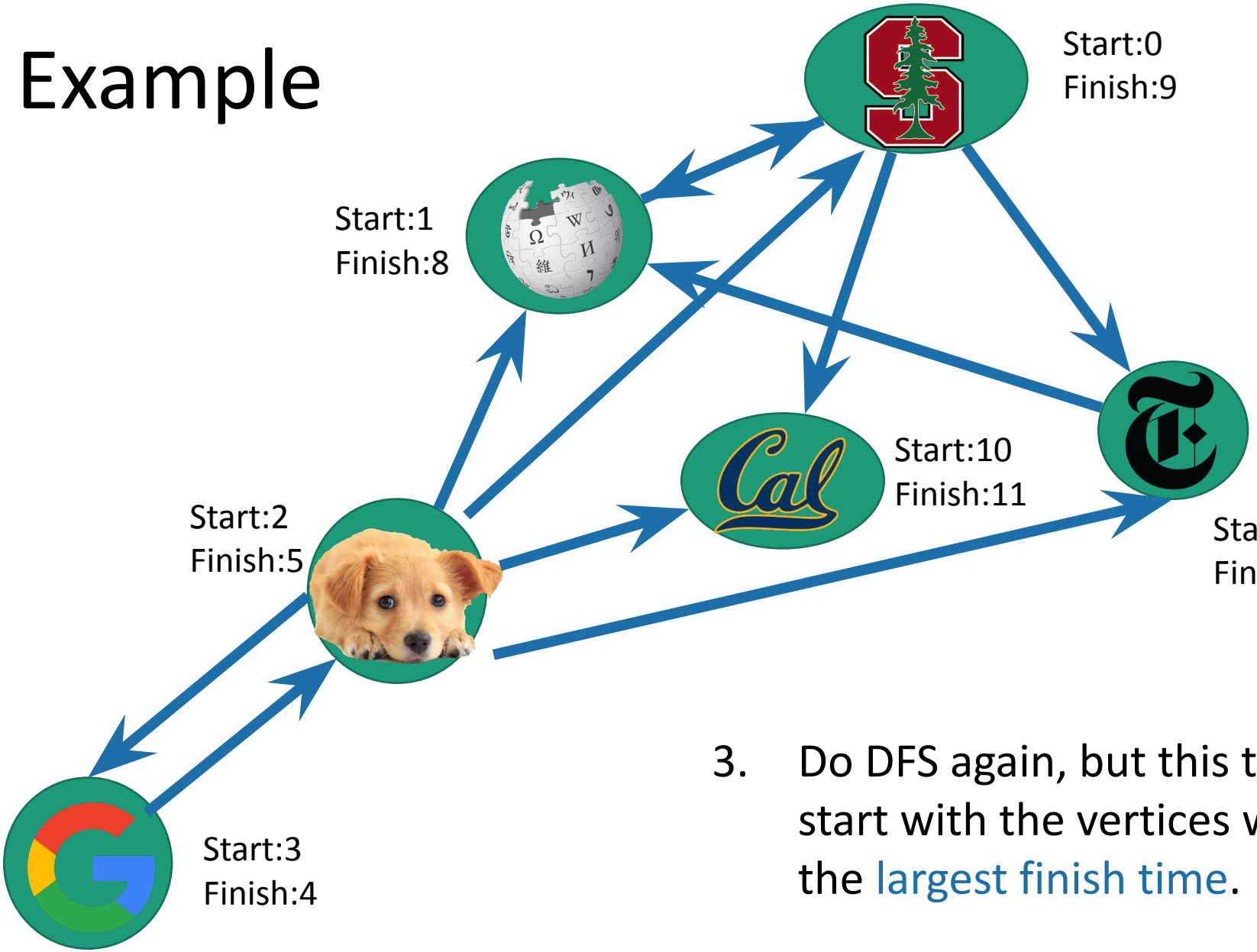
2. Reverse all the edges.

Example



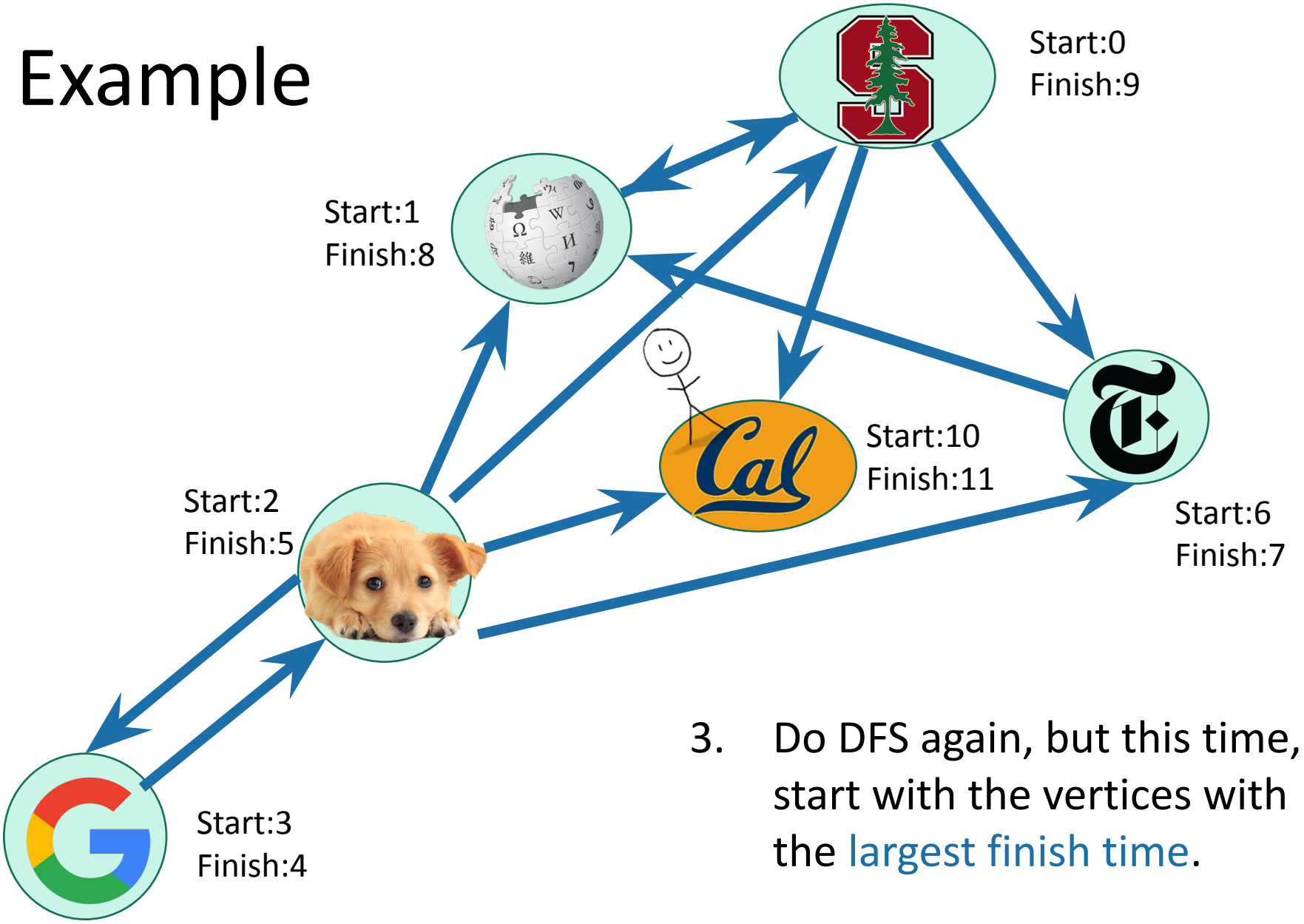
2. Reverse all the edges.

Example



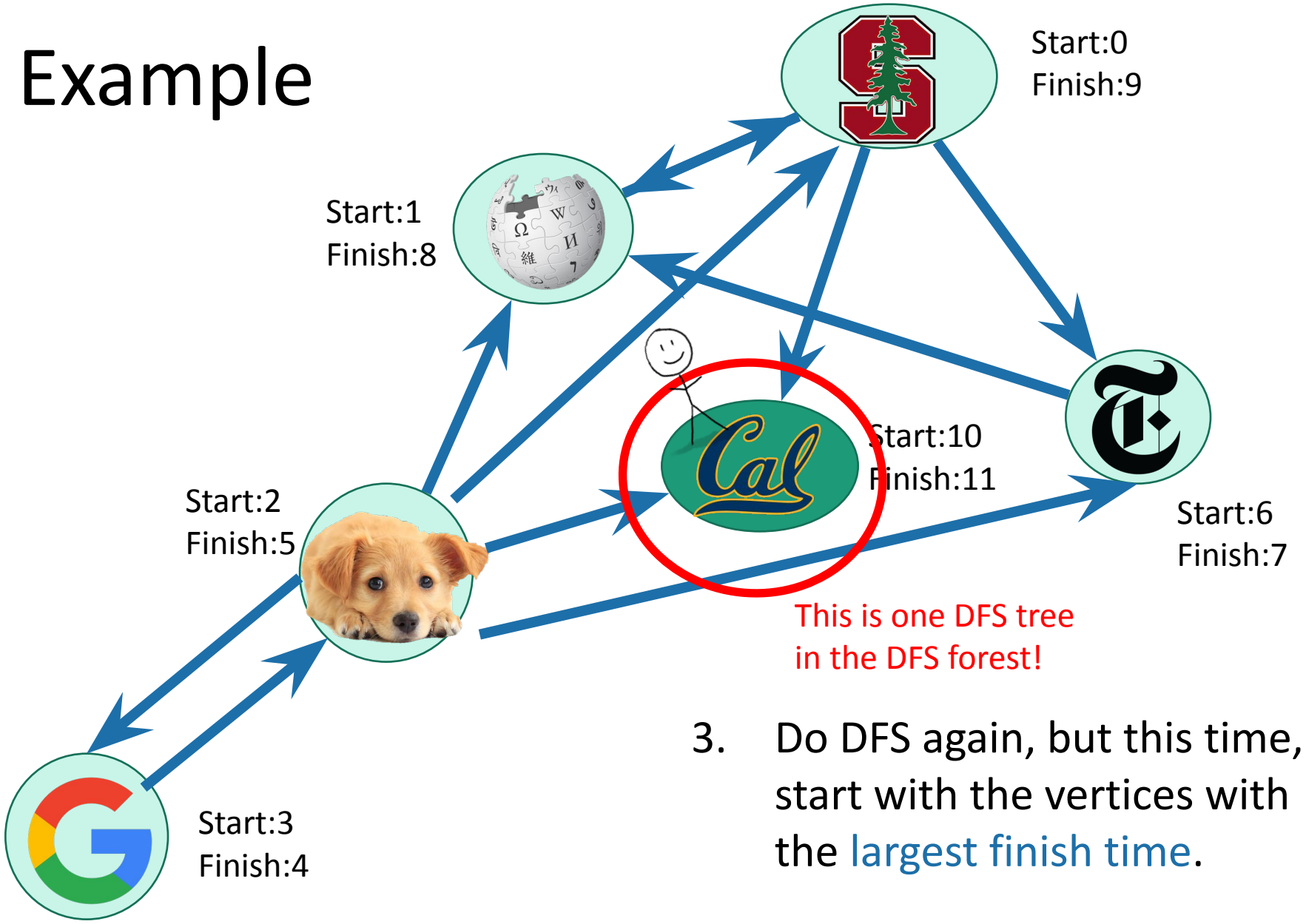
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



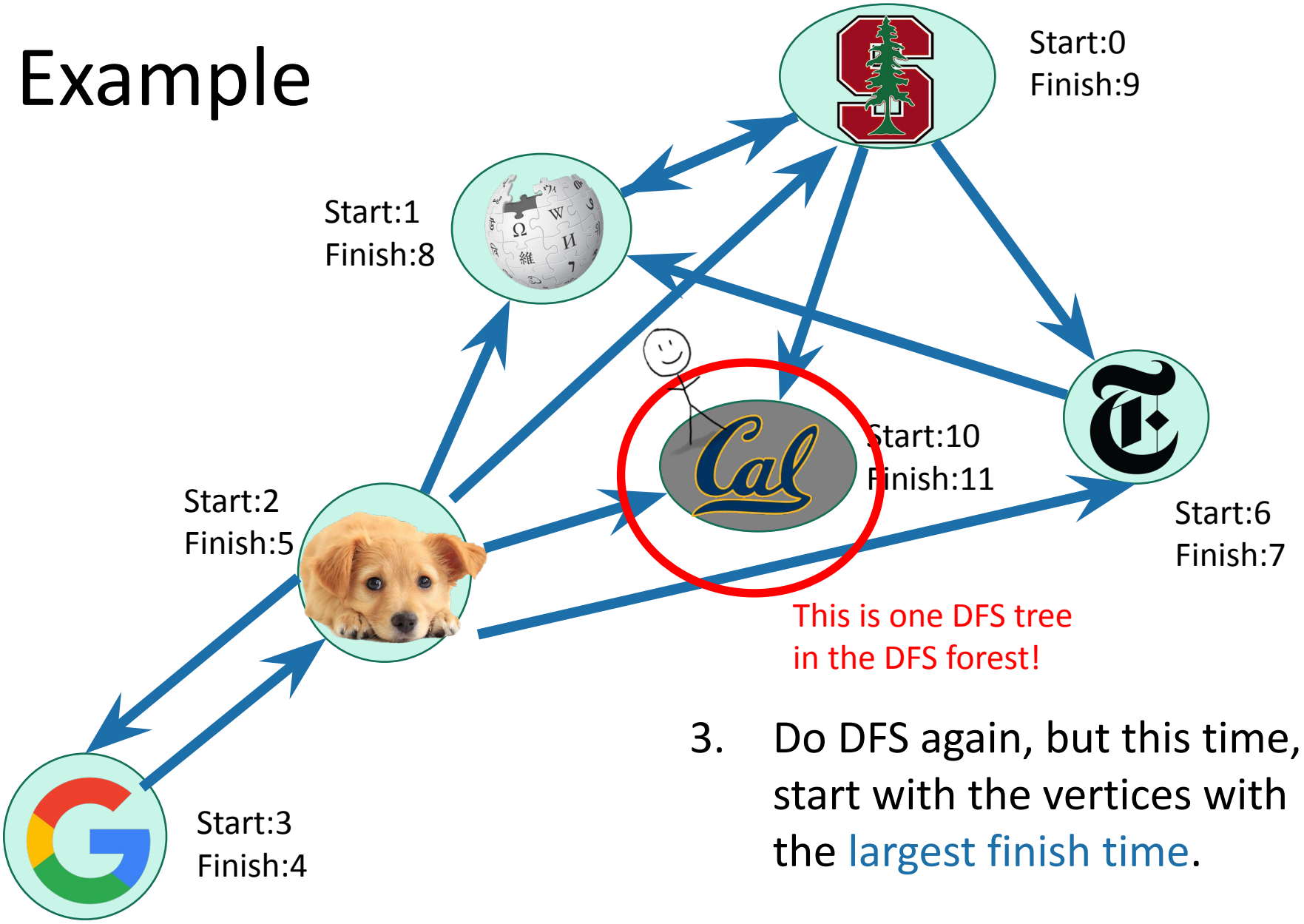
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



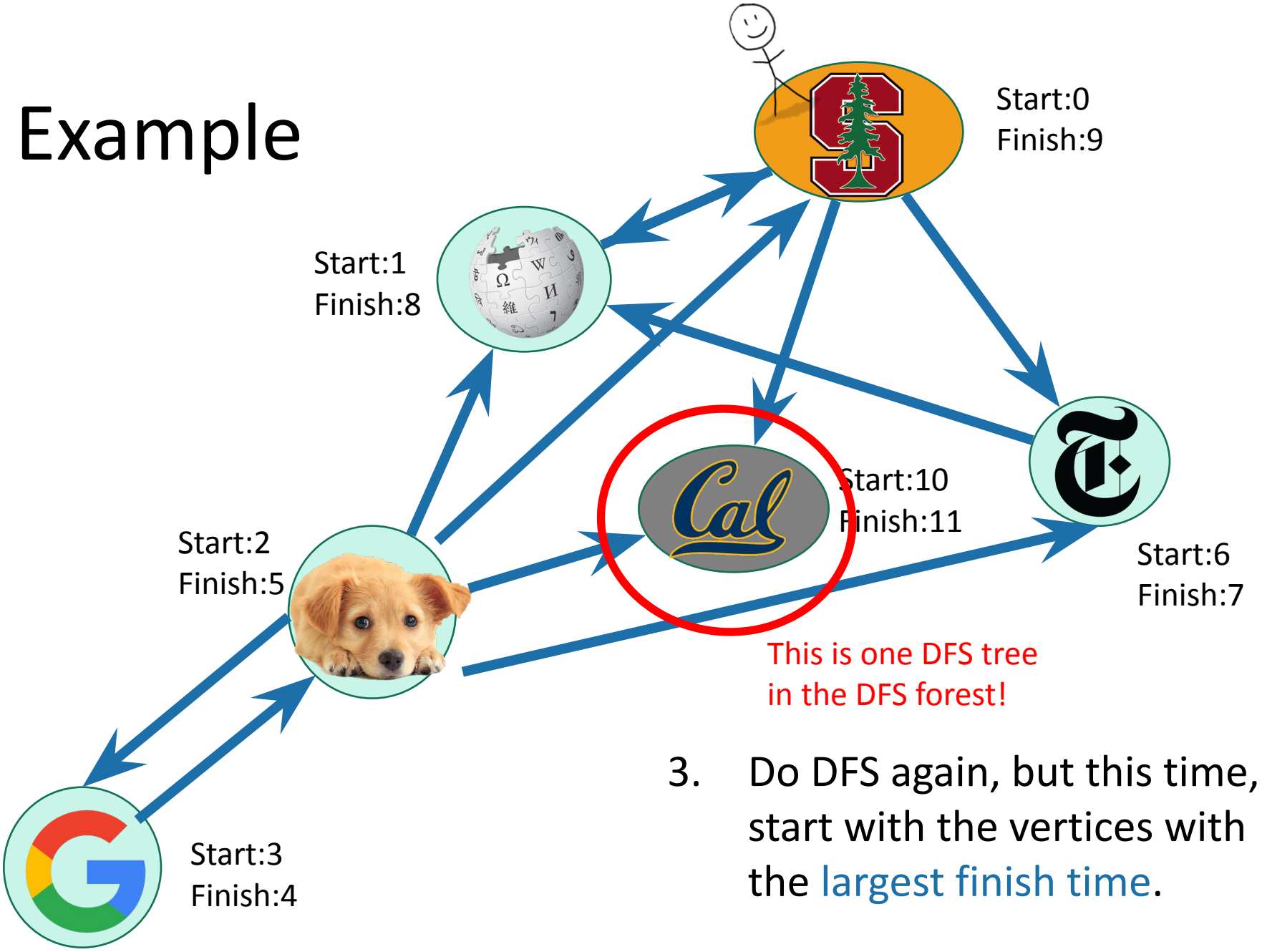
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



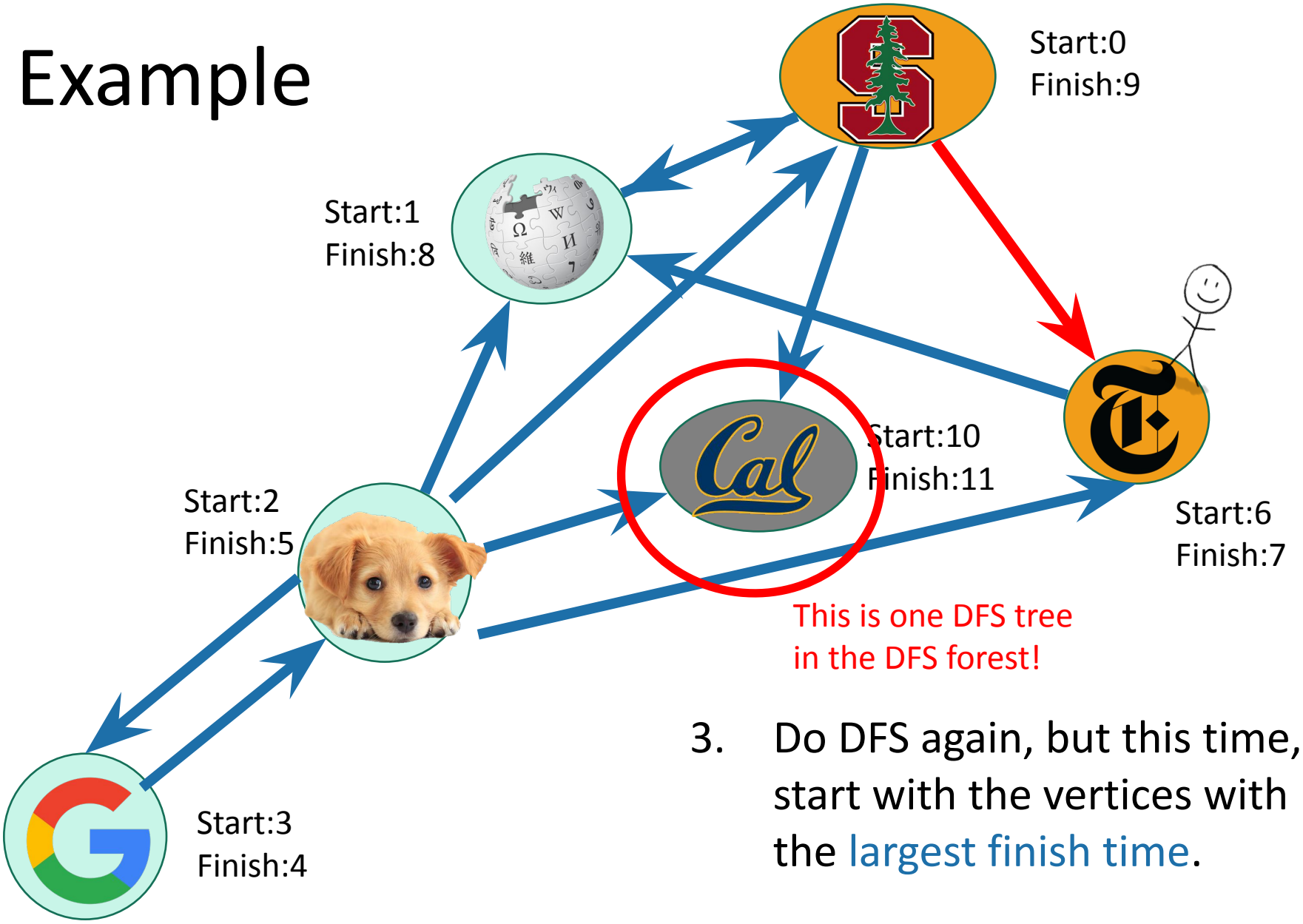
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



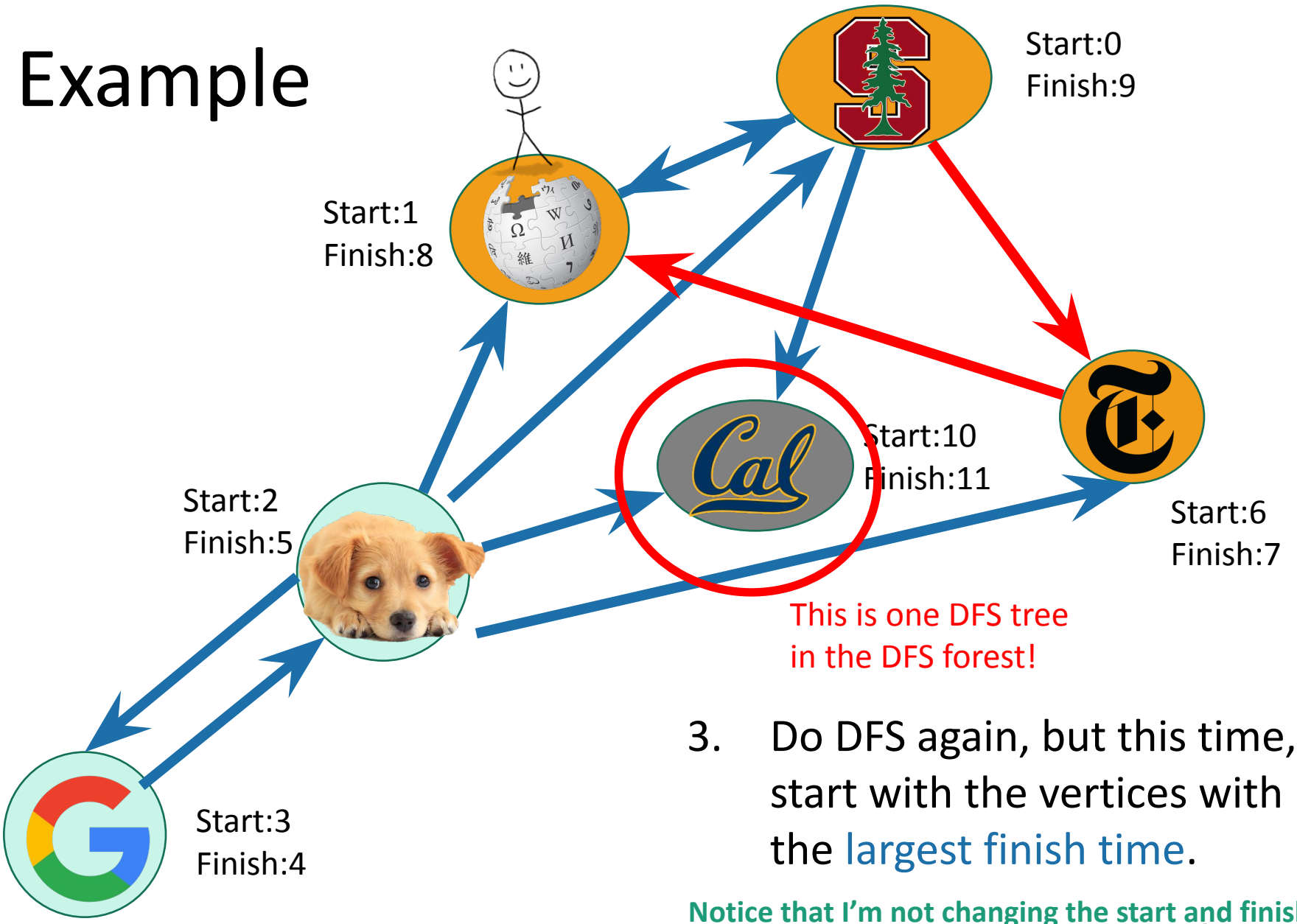
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



3. Do DFS again, but this time, start with the vertices with the largest finish time.

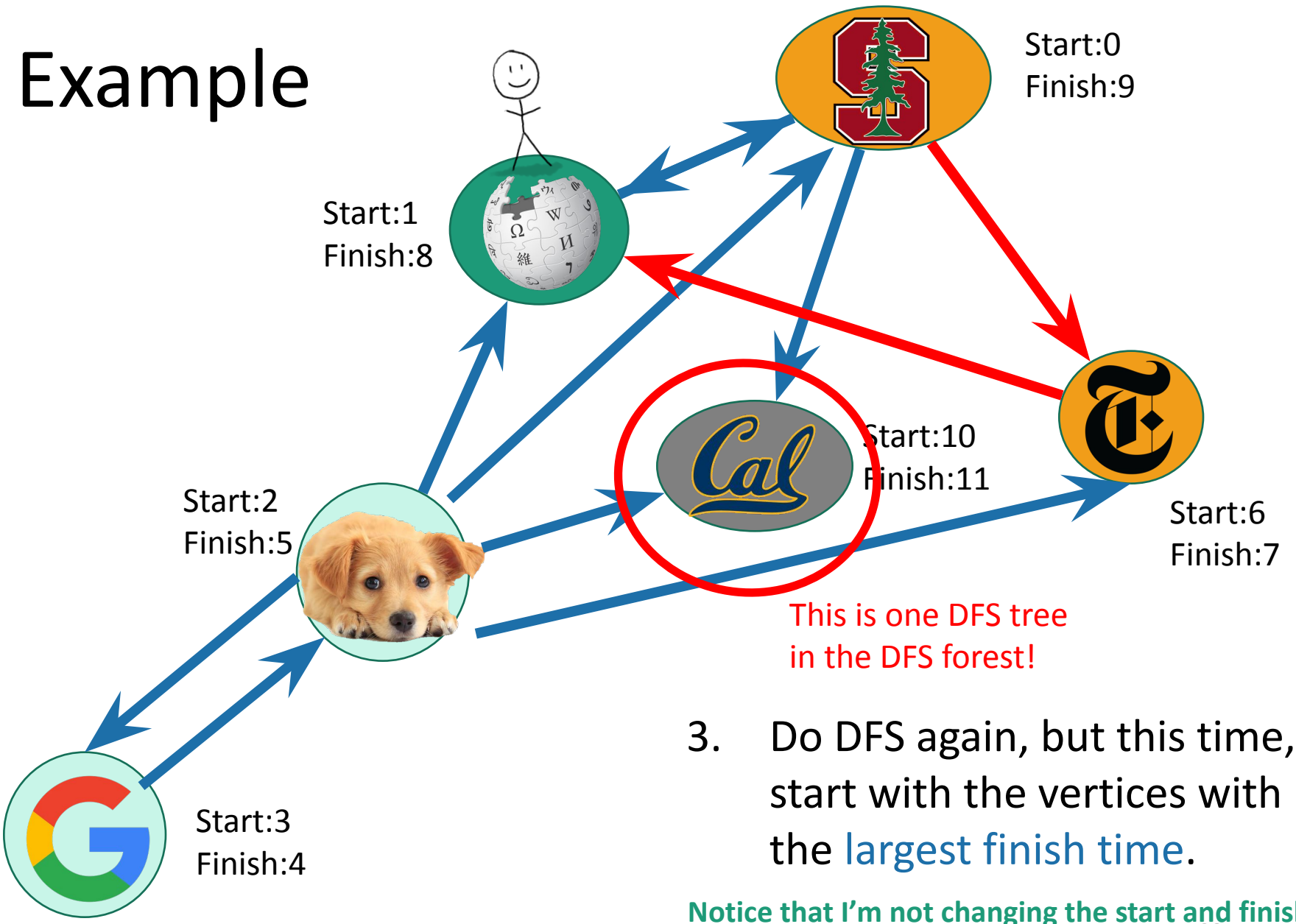
Example



3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

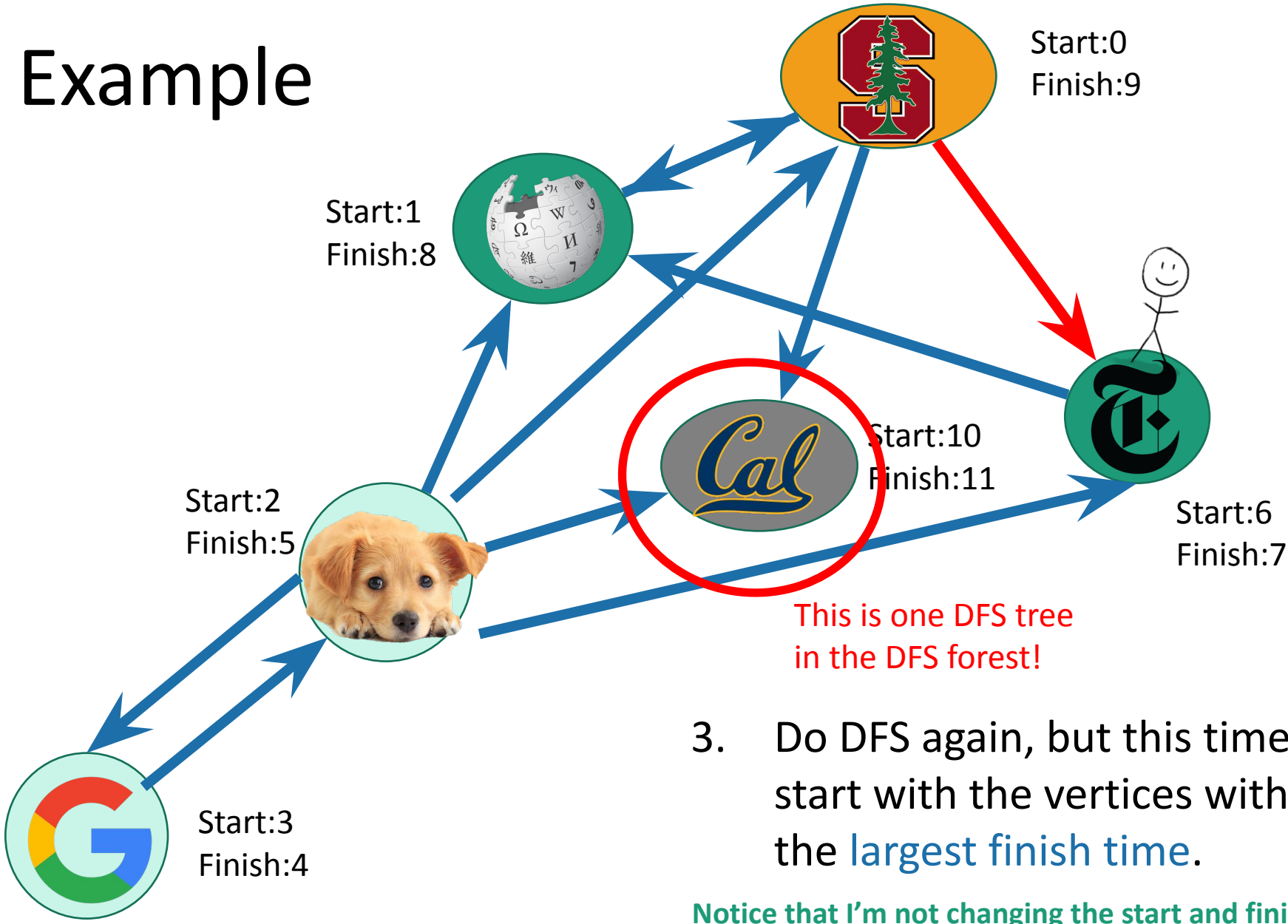
Example



3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

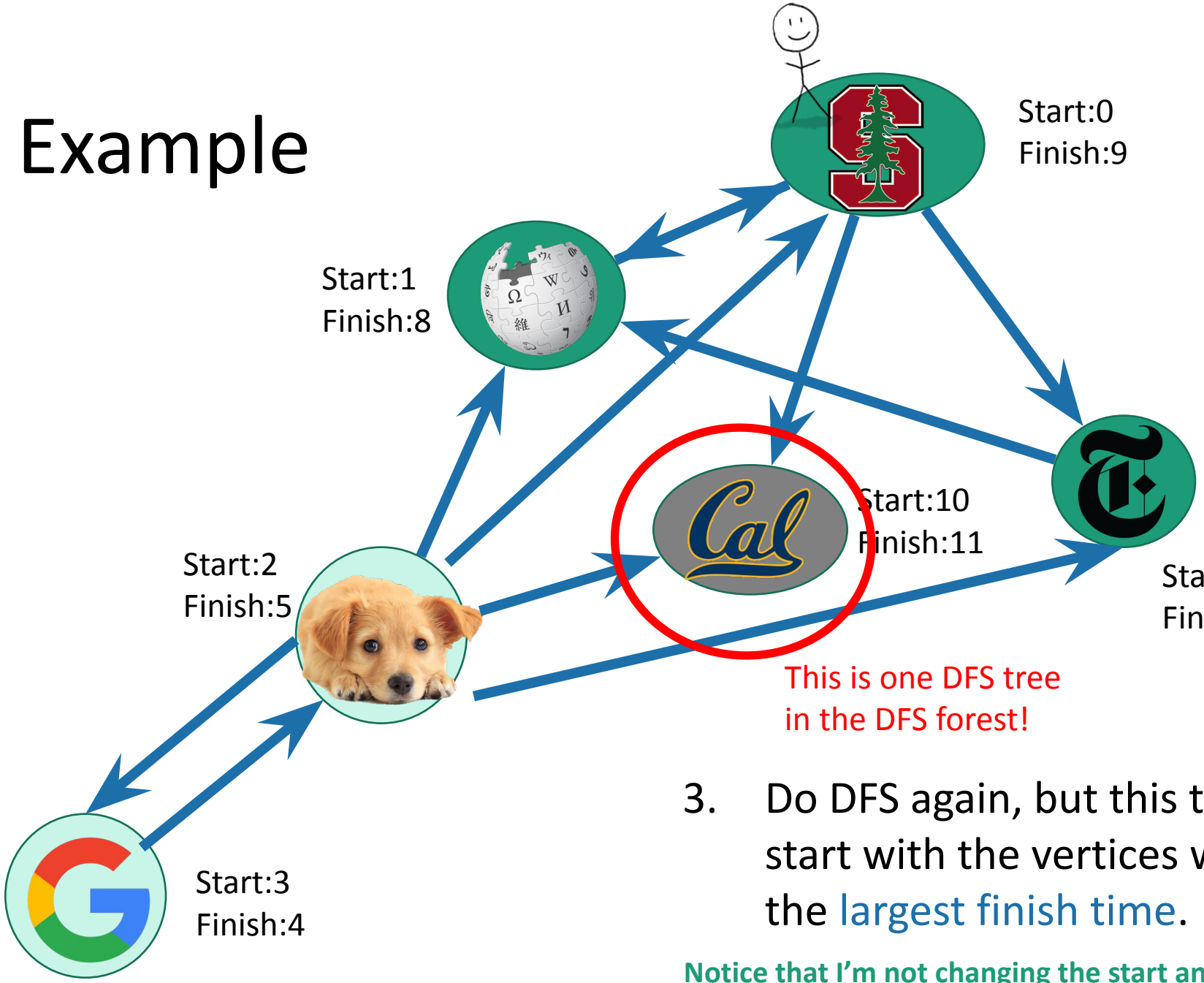
Example



3. Do DFS again, but this time, start with the vertices with the largest finish time.

Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

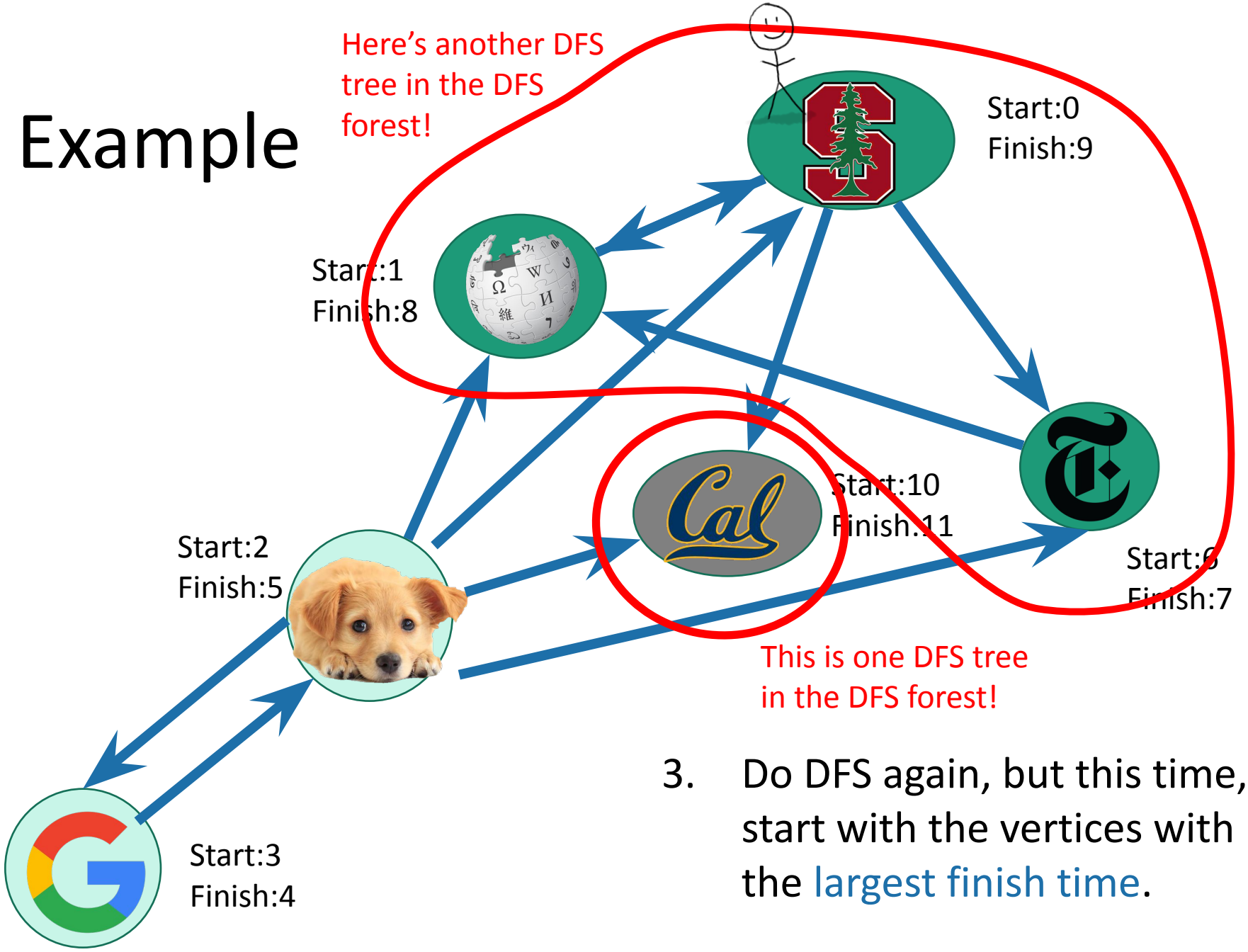
Example



3. Do DFS again, but this time, start with the vertices with the largest finish time.

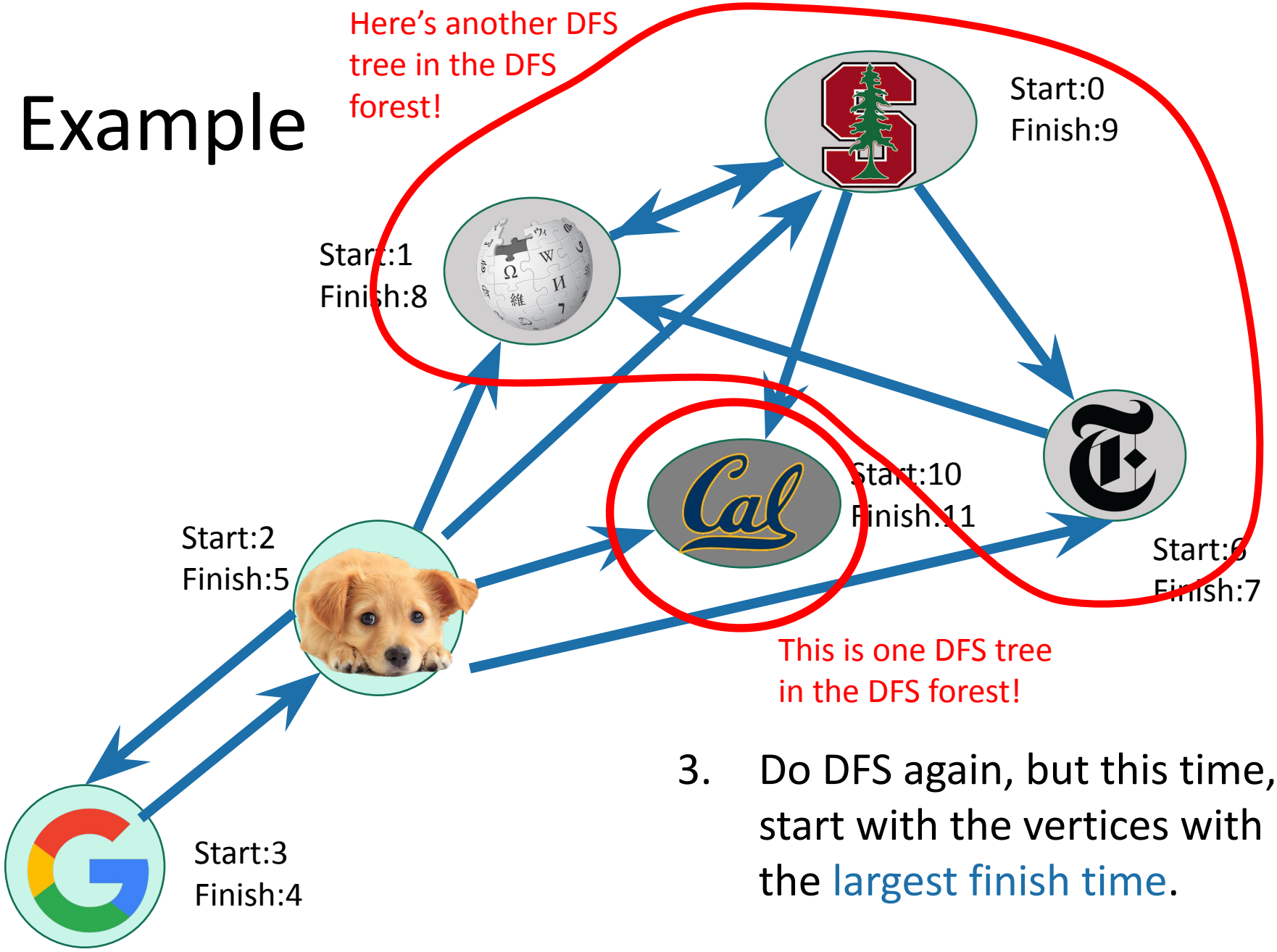
Notice that I'm not changing the start and finish times – I'm keeping them from the first run.

Example



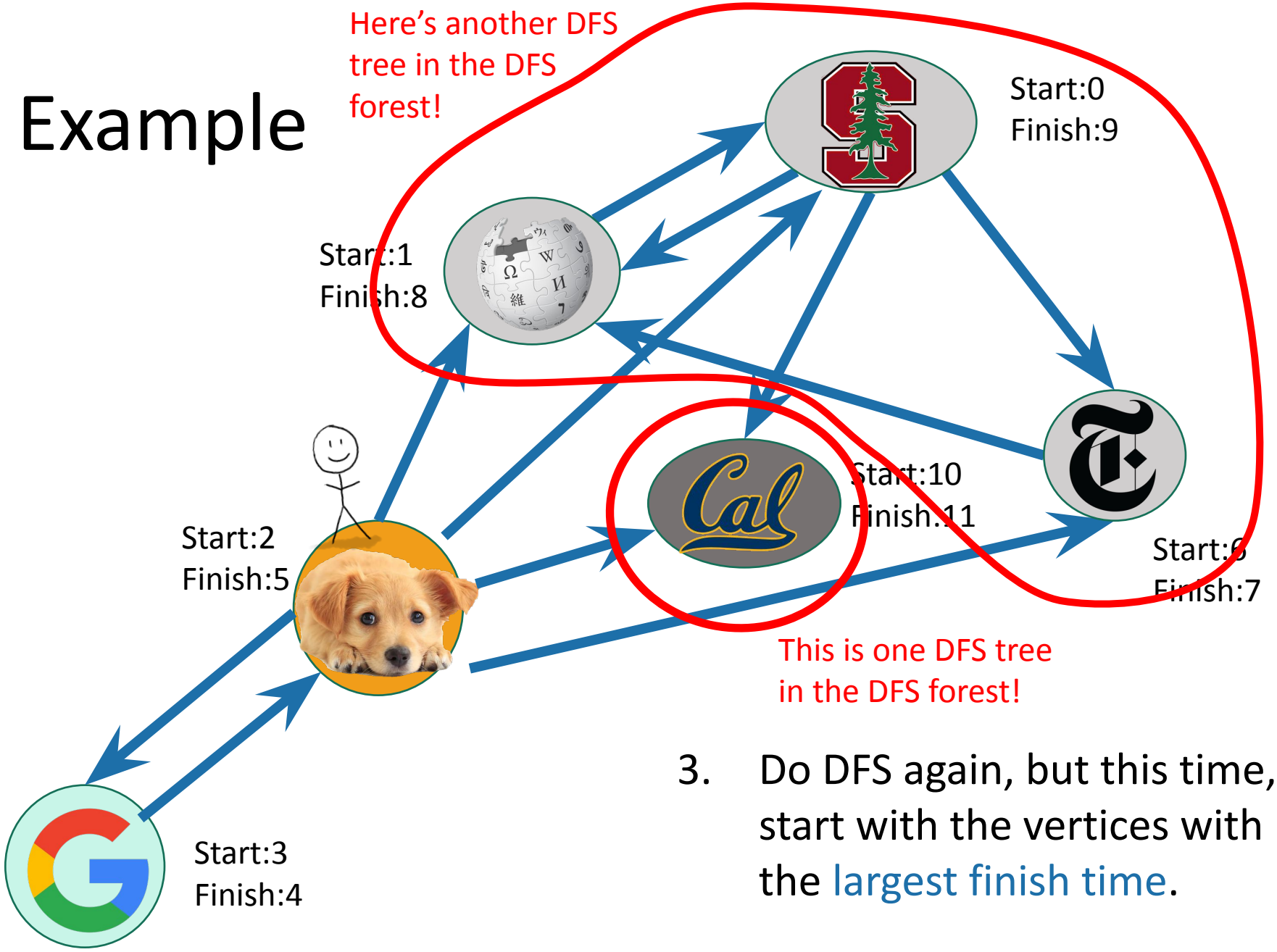
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



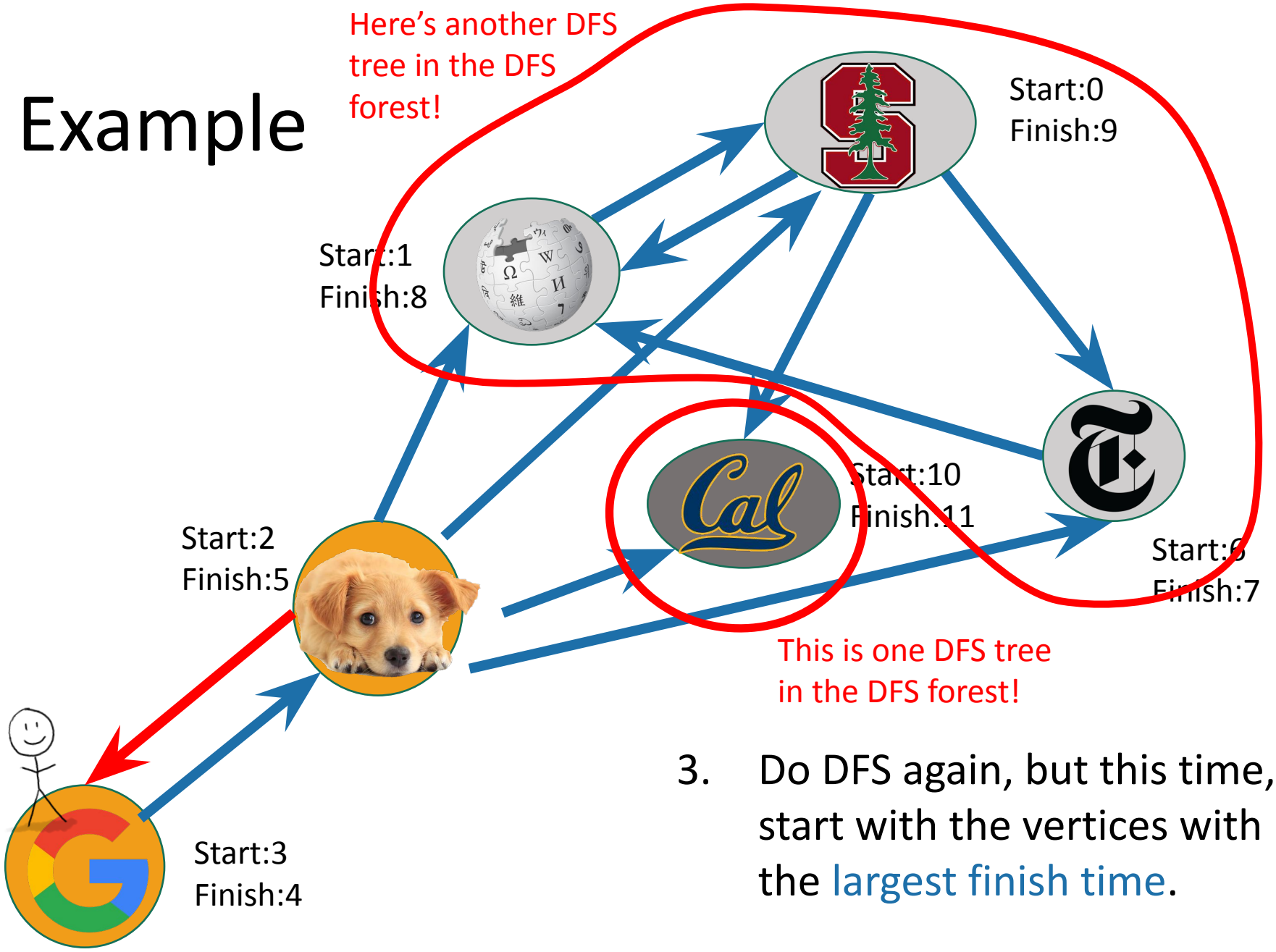
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



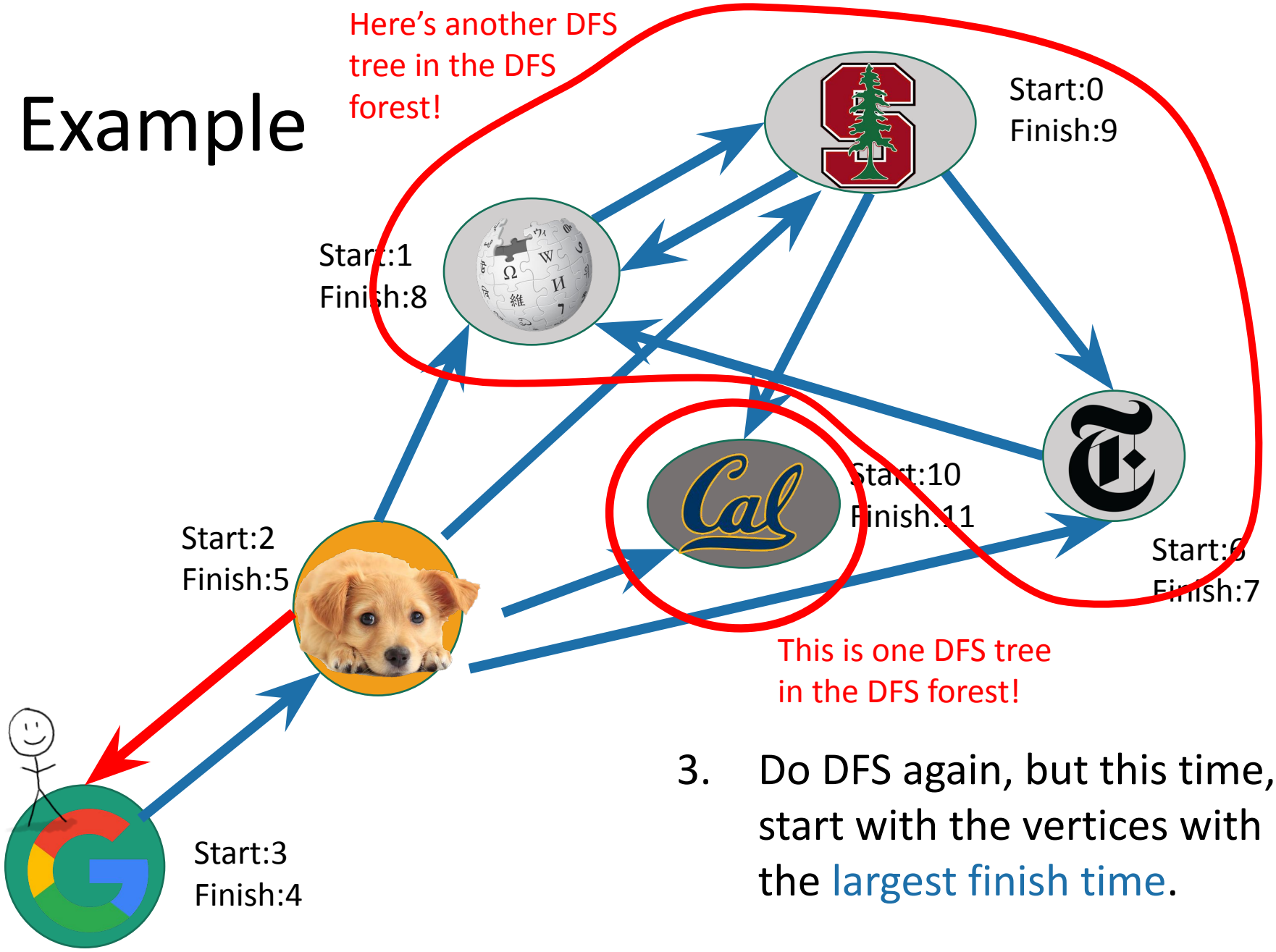
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



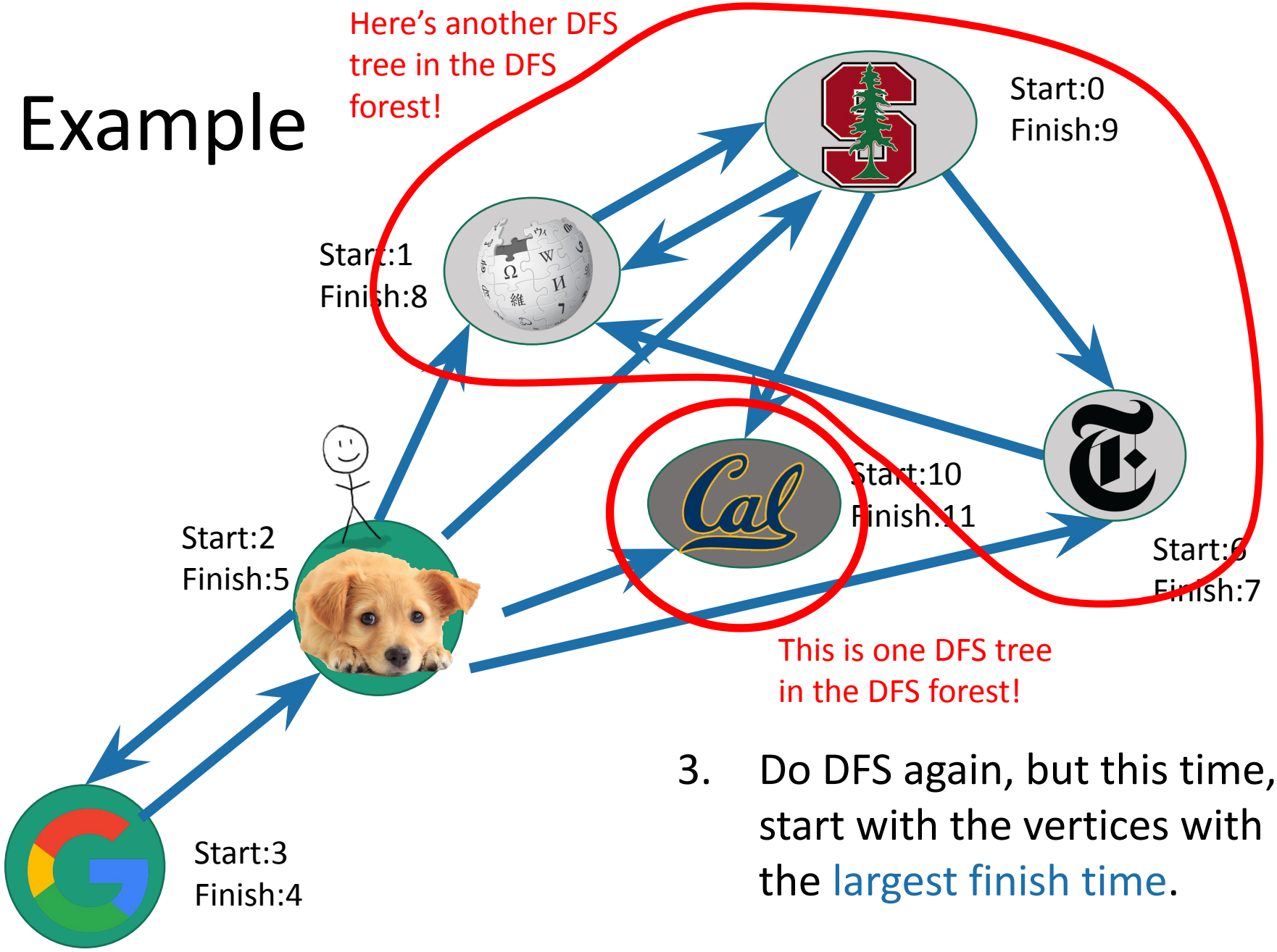
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



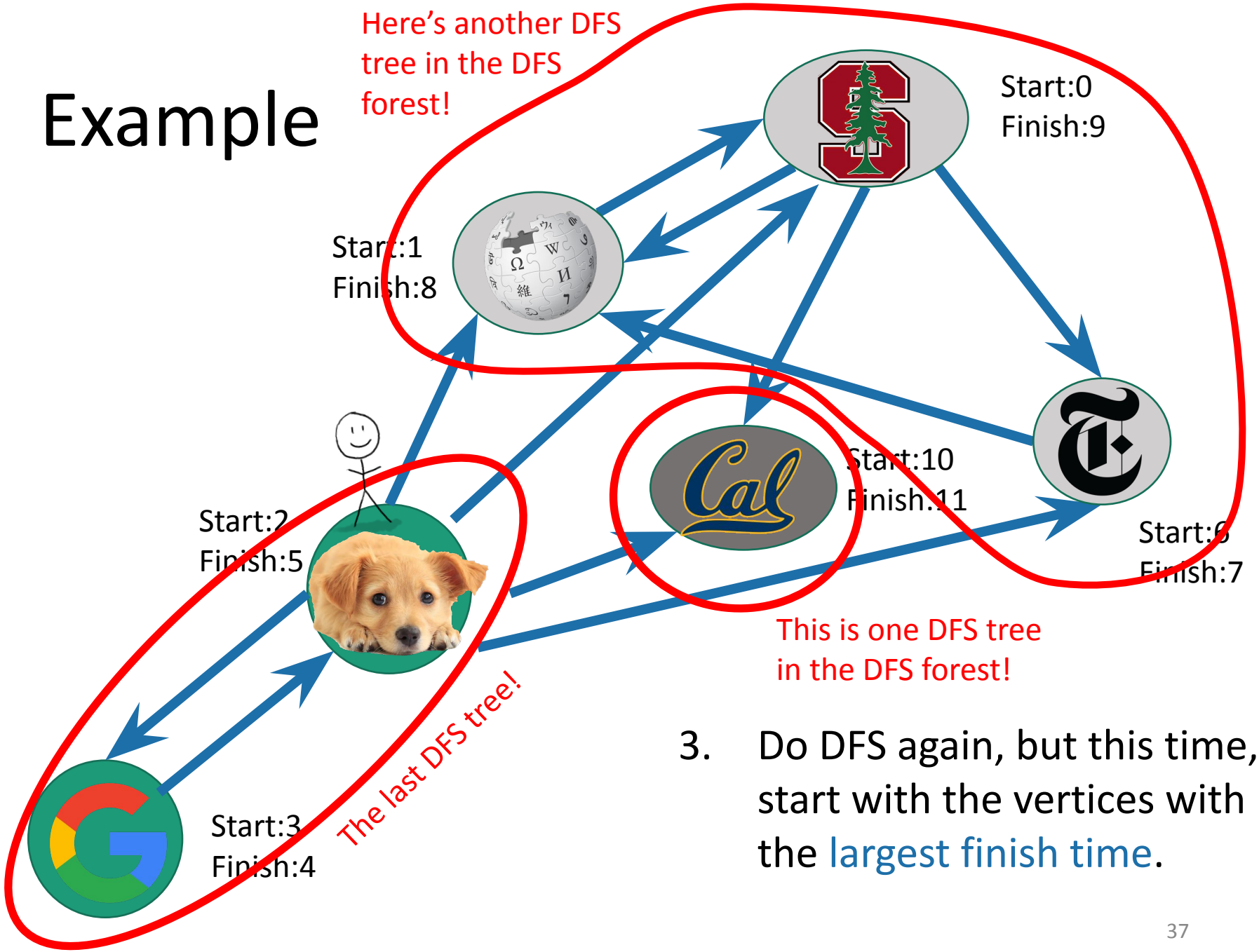
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



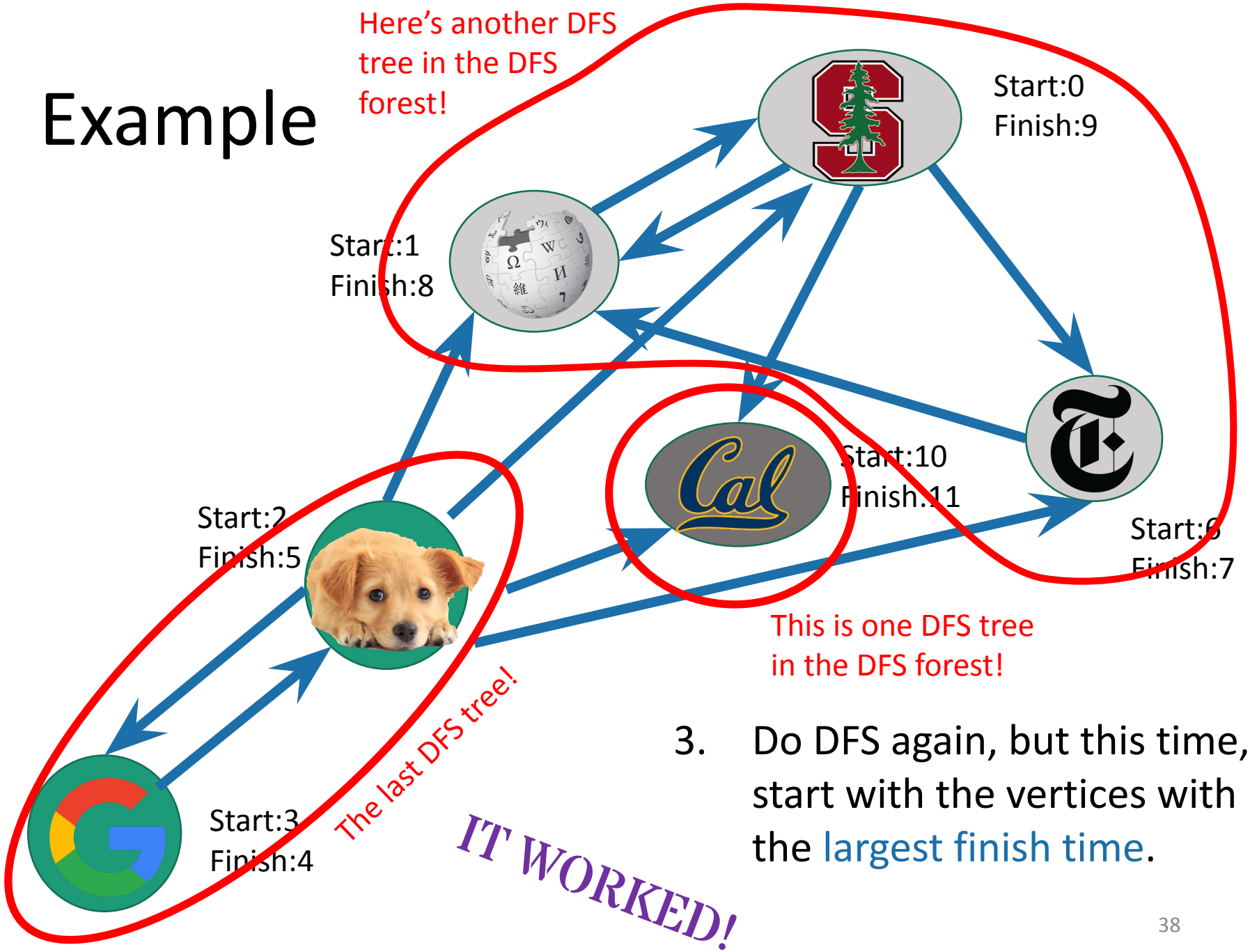
3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



3. Do DFS again, but this time, start with the vertices with the largest finish time.

Example



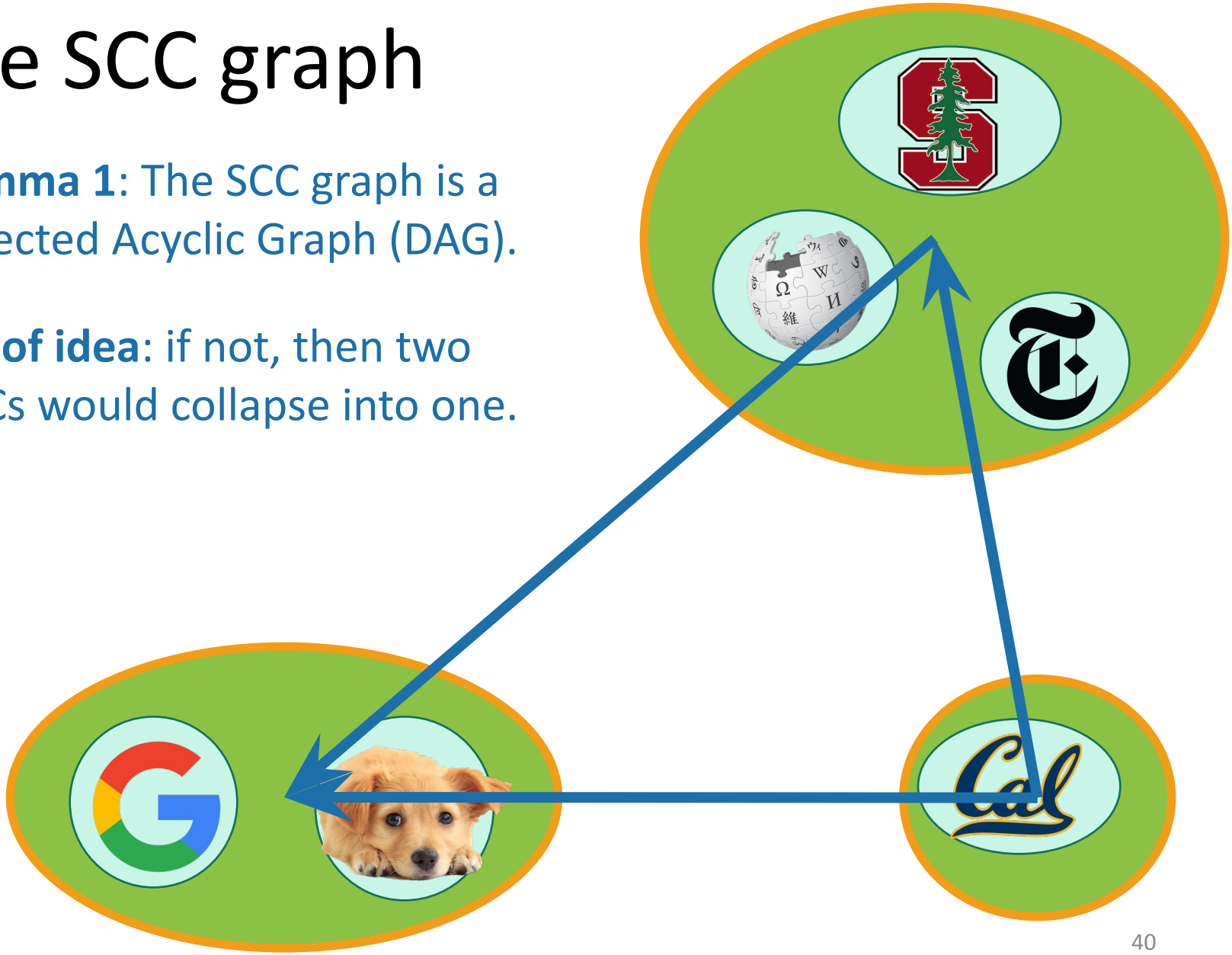
3. Do DFS again, but this time, start with the vertices with the largest finish time.

One question

The SCC graph

Lemma 1: The SCC graph is a Directed Acyclic Graph (DAG).

Proof idea: if not, then two SCCs would collapse into one.

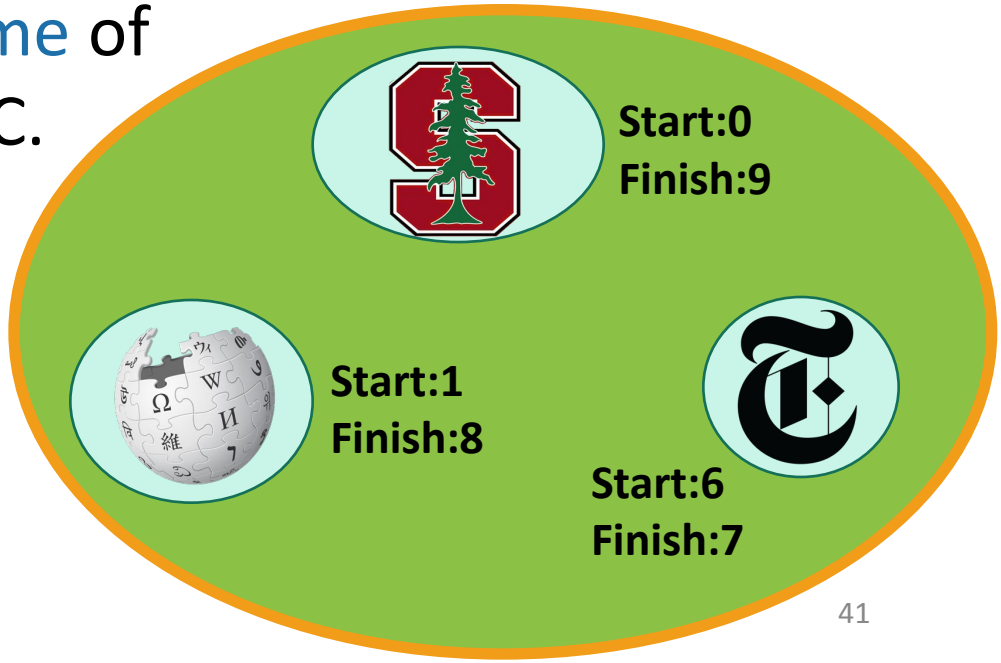


all times are with respect to the **first** DFS run

Starting and finishing times in a SCC

Definitions:

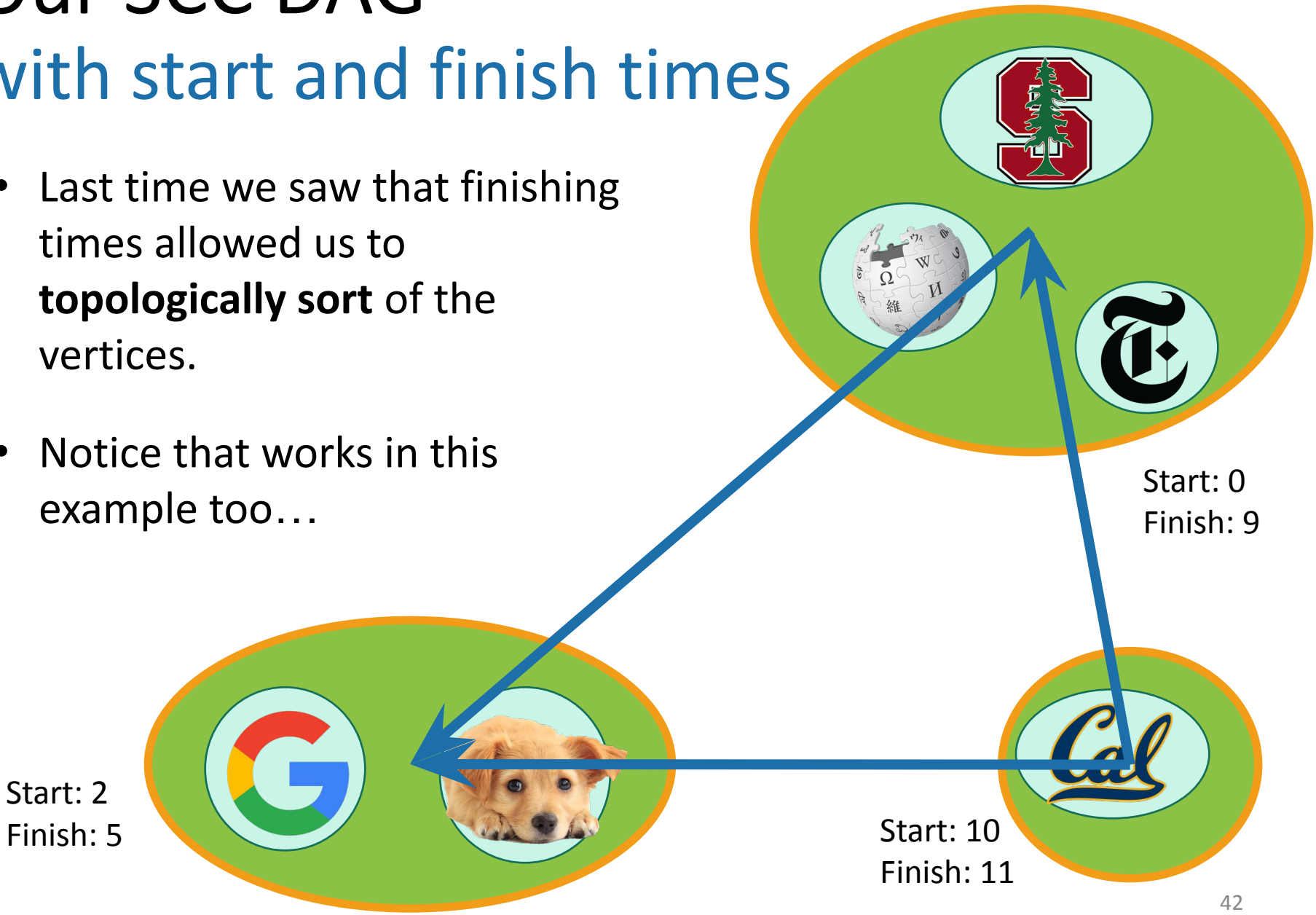
- The **finishing time** of a SCC is the largest finishing time of any element of that SCC.
- The **starting time** of a SCC is the smallest starting time of any element of that SCC.



Start: 0
Finish: 9

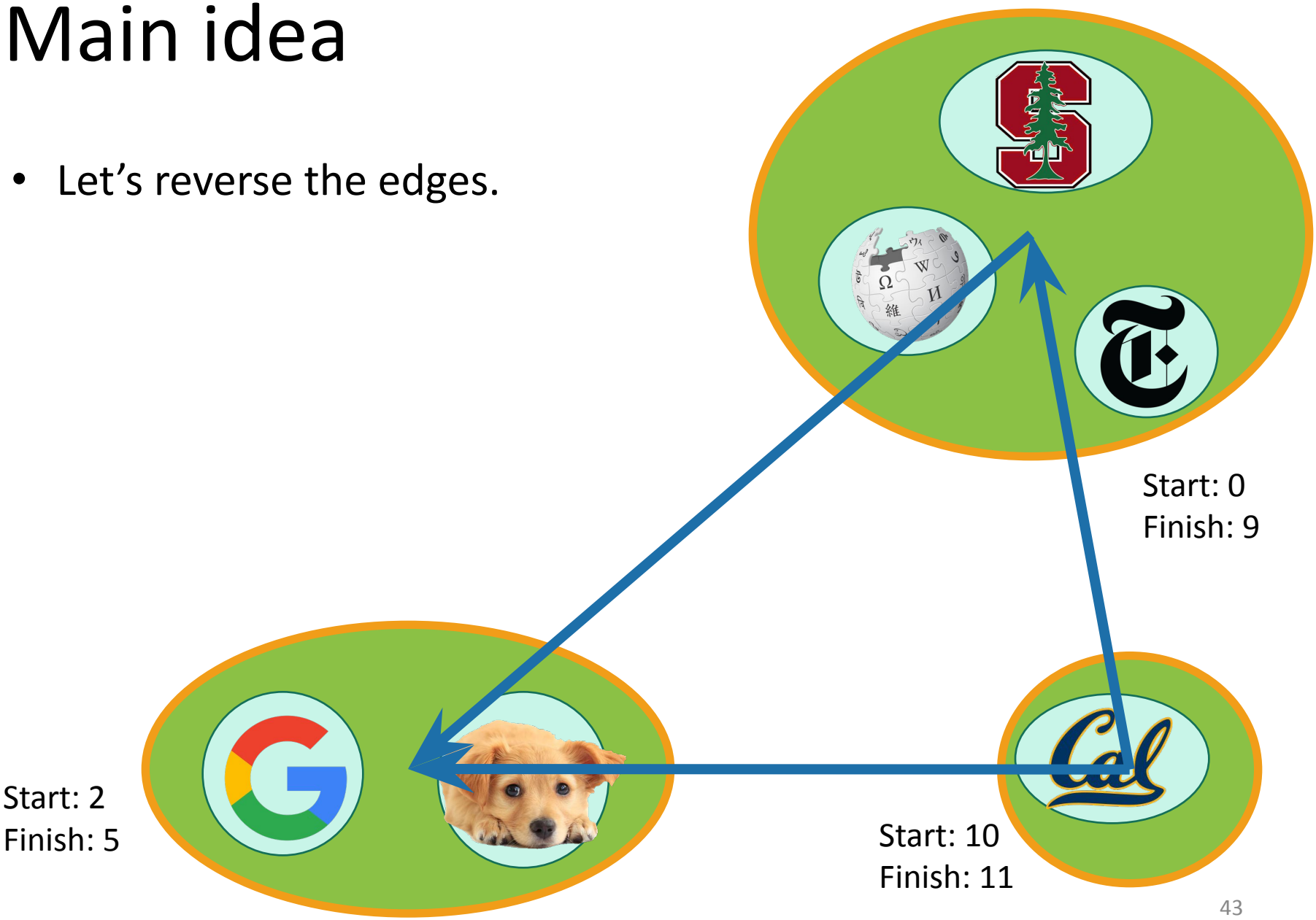
Our SCC DAG
with start and finish times

- Last time we saw that finishing times allowed us to **topologically sort** of the vertices.
- Notice that works in this example too…



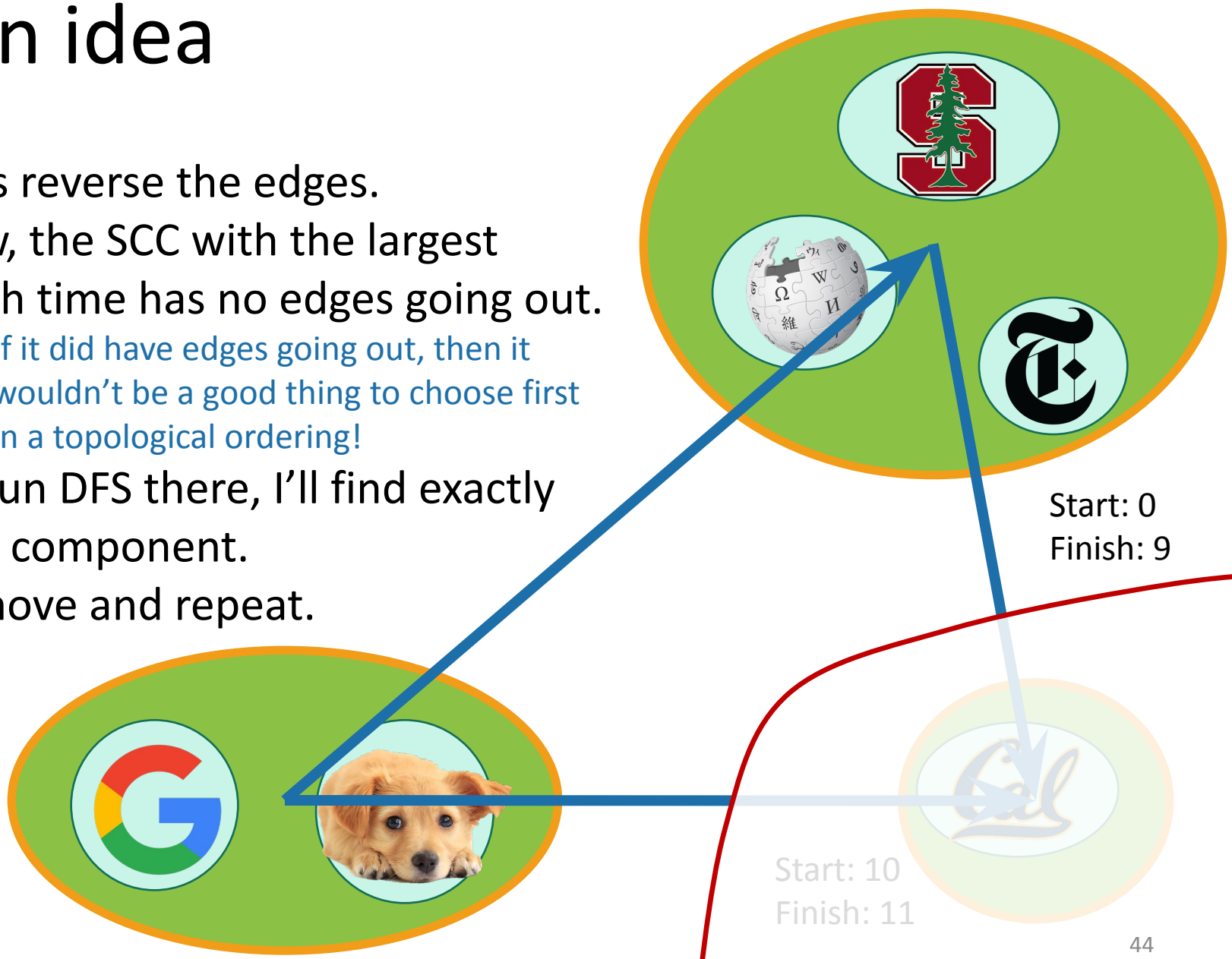
Main idea

- Let's reverse the edges.



Main idea

- Let's reverse the edges.
- Now, the SCC with the largest finish time has no edges going out.
  - If it did have edges going out, then it wouldn't be a good thing to choose first in a topological ordering!
- If I run DFS there, I'll find exactly that component.
- Remove and repeat.



Let’s make this idea formal.

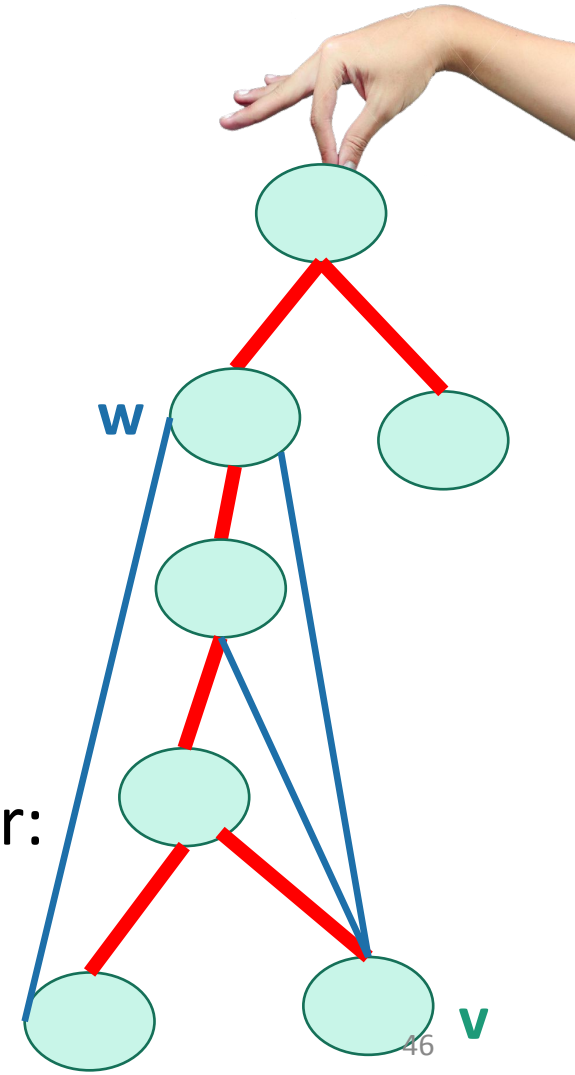
Recall

- If v is a descendant of w in this tree:

- If w is a descendant of v in this tree:

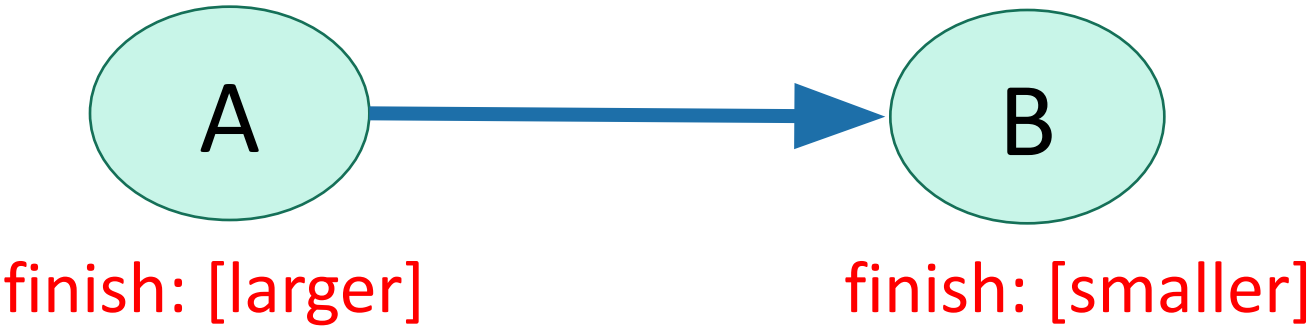
- If neither are descendants of each other:

(or the other way around)



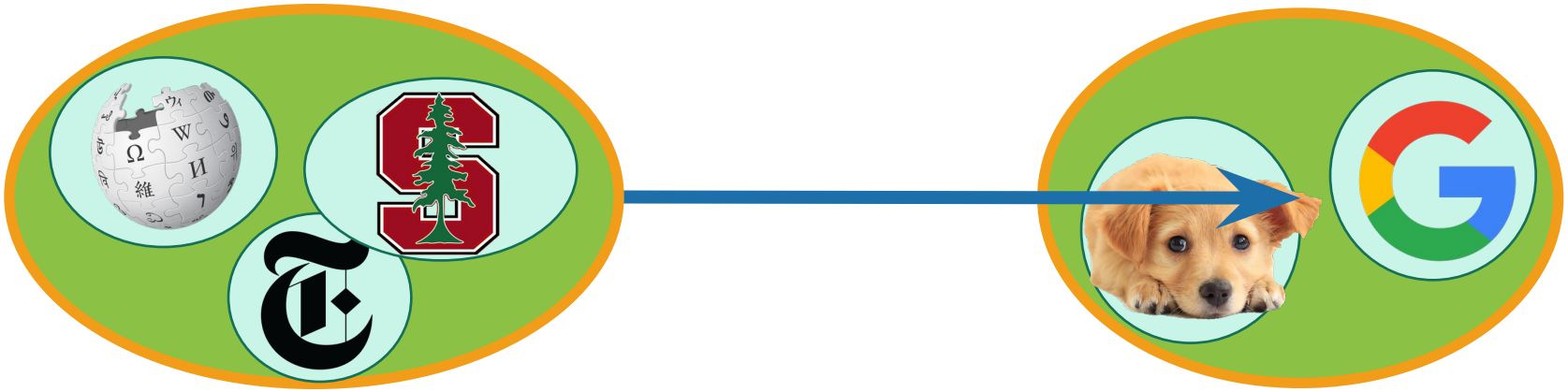
As we mentioned earlier:

Claim: In a DAG, we'll always have:



Same thing, in the SCC DAG.

- Claim: we'll always have



finish: [larger]

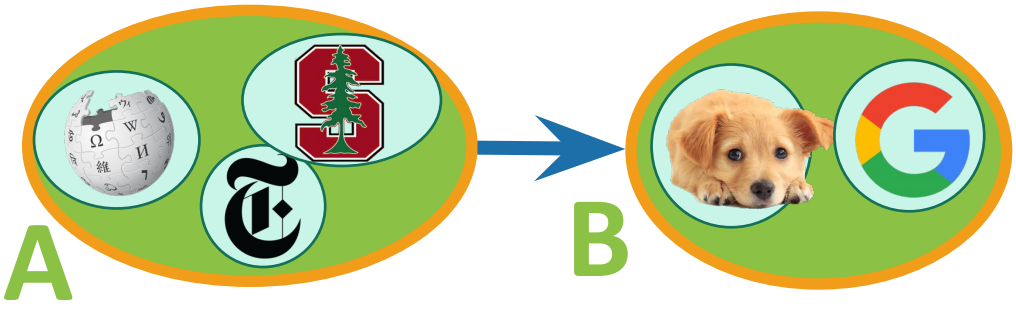
finish: [smaller]

Let's call it Lemma 2

- If there is an edge like this:

- Then A.finish > B.finish.

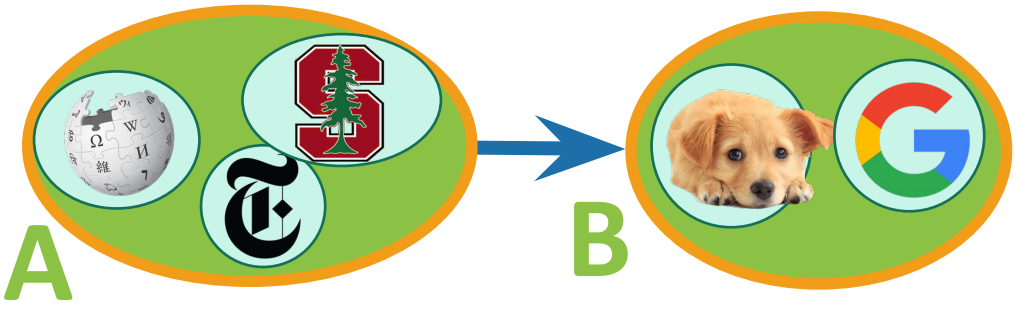
Proof idea



Want to show A.finish > B.finish.

- **Two cases**:
  - We reached **A** before **B** in our first DFS.
  - We reached **B** before **A** in our first DFS.

Proof idea



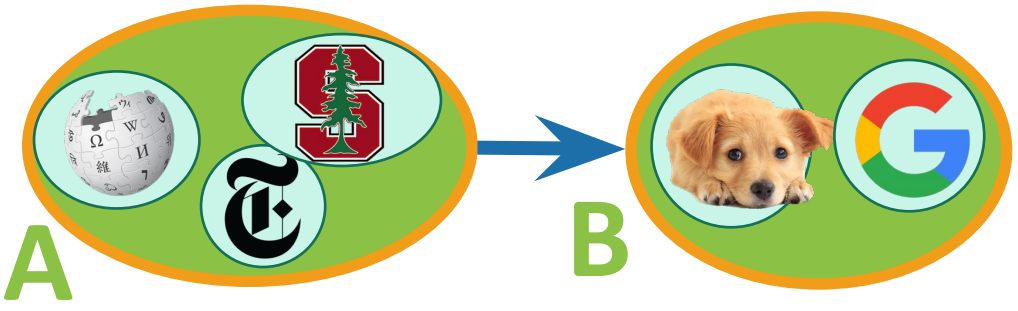
Want to show A.finish > B.finish.

- **Case 1**: We reached **A** before **B** in our first DFS.
- Say that:
  - **y** has the largest finish in **B**; B.finish = y.finish
  - **z** was discovered first in **A**; A.finish >= z.finish
- Then:
  - Reach **A** before **B**
  - => we will discover **y** via **z**
  - => **y** is a descendant of **z** in the DFS forest.

- Then

aka,
A.finish > B.finish

Proof idea



Want to show A.finish > B.finish.

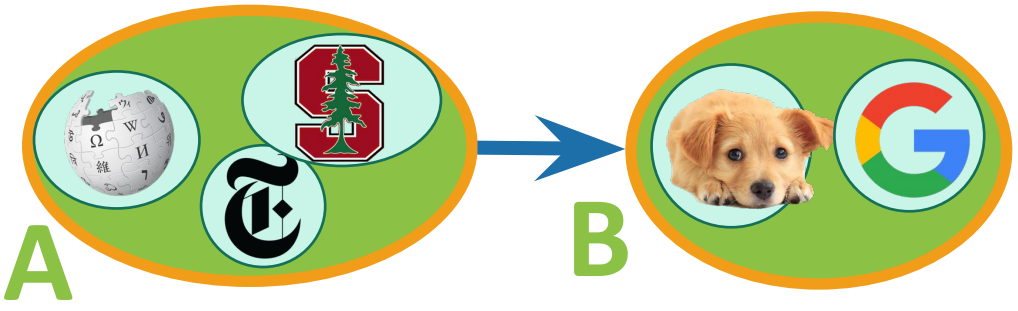
- **Case 2**: We reached **B** before **A** in our first DFS.

- There are no paths from B to A
  - because the SCC graph has no cycles

- So we completely finish exploring B and never reach A.
- A is explored later after we restart DFS.

aka,
A.finish > B.finish

Proof idea



Want to show A.finish > B.finish.

- **Two cases**:
  - We reached **A** before **B** in our first DFS.
  - We reached **B** before **A** in our first DFS.

- In either case:

A.finish > B.finish

which is what we wanted to show.

This establishes:

Lemma 2

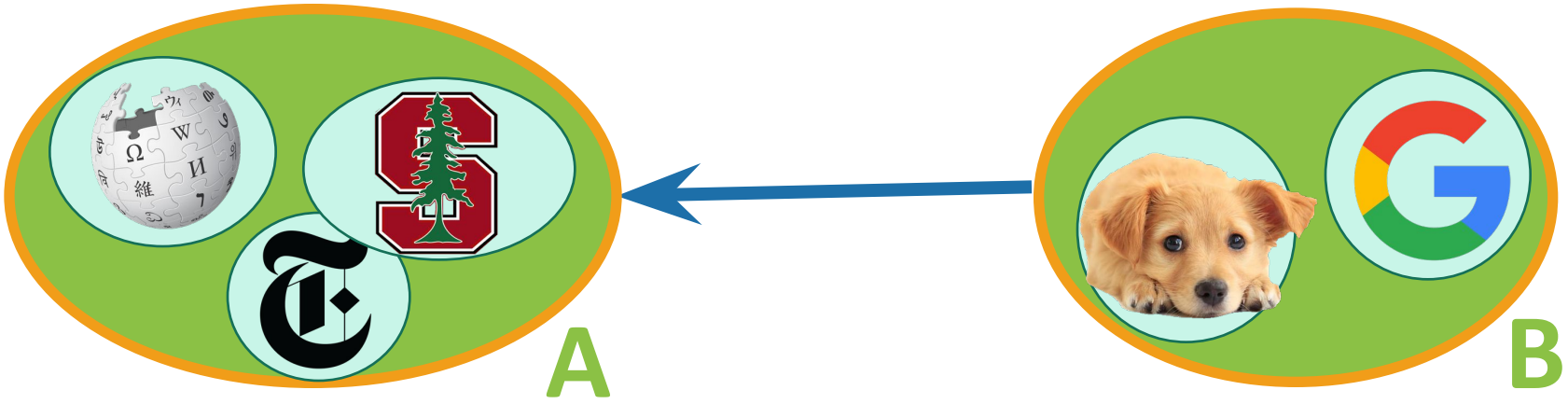
- If there is an edge like this:

- Then A.finish > B.finish.

This establishes:

Corollary 1

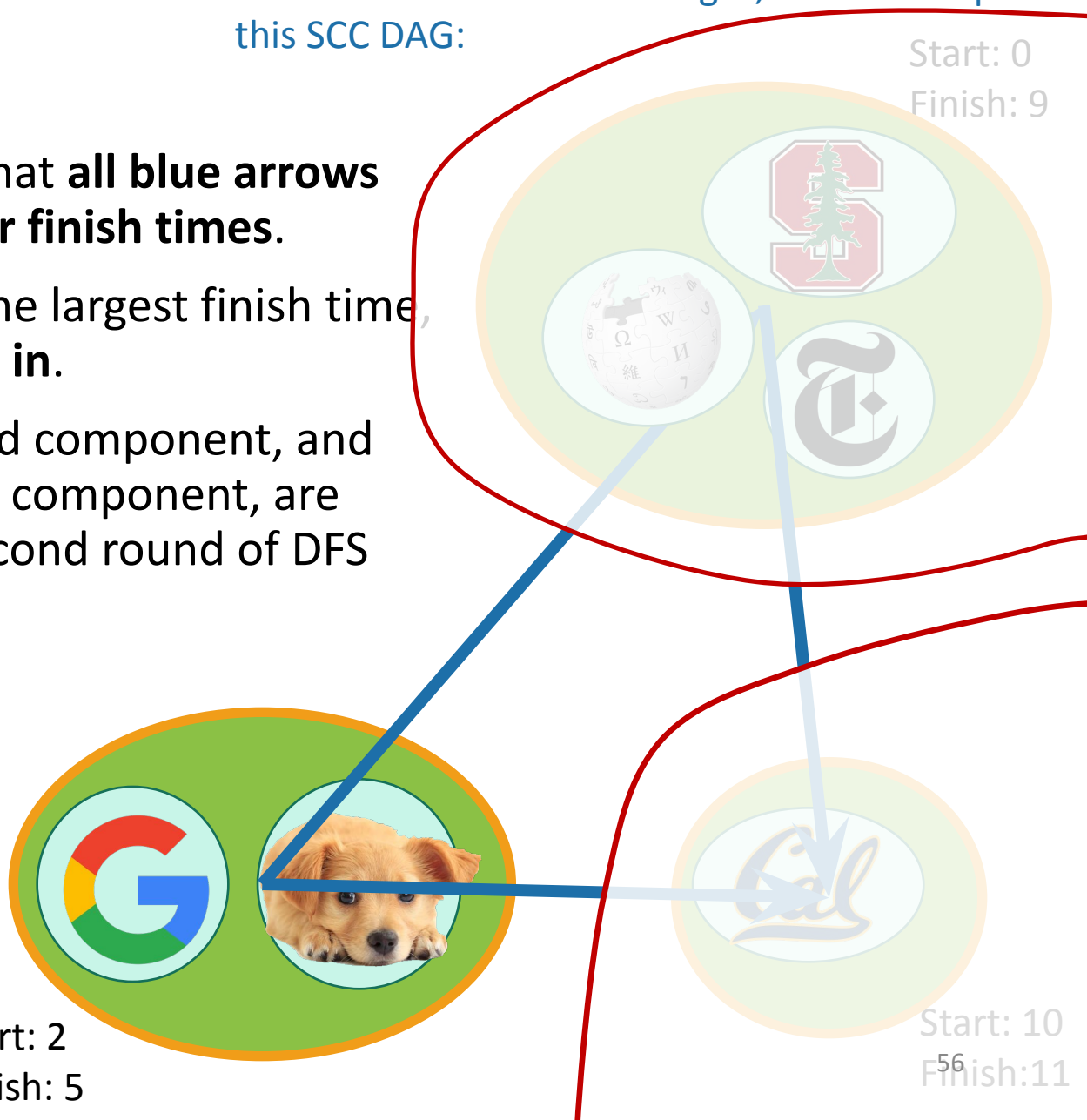
- If there is an edge like this in the **reversed graph**:



- Then A.finish > B.finish.

Now we see why this finds SCCs.

Remember that after the first round of DFS, and after we reversed all the edges, we ended up with this SCC DAG:



- The Corollary says that **all blue arrows point towards larger finish times**.
- So if we start with the largest finish time, **all blue arrows lead in**.
- Thus, that connected component, and only that connected component, are reachable by the second round of DFS

- Now, we've deleted that first component.
- The next one has the **next biggest finishing time.**
- So **all remaining blue arrows lead in.**
- Repeat.

Formally, we prove it by induction

- **Theorem**: The algorithm we saw before will correctly identify strongly connected components.

- **Inductive hypothesis**:
  - The first t trees found in the second (reversed) DFS forest are the t SCCs with the largest finish times.

- **Base case**: (t=0)
  - The first 0 trees found in the reversed DFS forest are the 0 SCCs with the largest finish times. **(TRUE)**

Inductive step

- **Assume by induction that the first t trees are the last-finishing SCCs.**
- Consider the $(t+1)^{st}$ tree produced, suppose the root is **x**.
- Suppose that **x** lives in the SCC **A**.
- Then **A**.finish > **B**.finish for all remaining SCCs **B**.
  - This is because we chose **x** to have the largest finish time.
- Then there are no edges leaving **A** in the remaining SCC DAG.
  - This follows from the Corollary.
- Then DFS started at **x** recovers exactly **A**.
  - It doesn't recover any more since nothing else is reachable.
  - It doesn't recover any less since A is strongly connected.
  - (Notice that we are using that A is still strongly connected when we reverse all the edges).
- **So the $(t+1)^{st}$ tree is the SCC with the $(t+1)^{st}$ biggest finish time.**

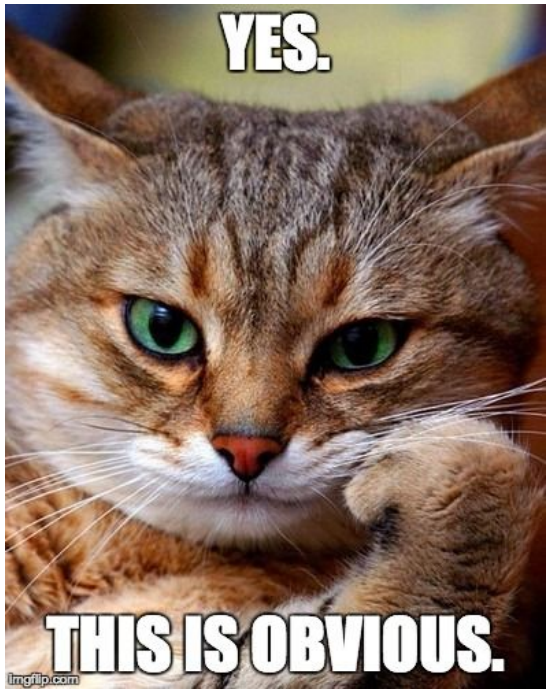
Formally, we prove it by induction

- **Theorem**: The algorithm we saw before will correctly identify strongly connected components.

- **Inductive hypothesis**:
  - The first t trees found in the second (reversed) DFS forest are the t SCCs with the largest finish times.

- **Base case**: *[done]*
- **Inductive step**: *[done]*
- **Conclusion:** The second (reversed) DFS forest contains all the SCCs as its trees!
  - (This is the **IH** when t = #SCCs)

Punchline:
we can find SCCs in time O(n + m)



`Algorithm:`

- Do DFS to create a **DFS forest**.
  - Choose starting vertices in any order.
  - Keep track of finishing times.
- Reverse all the edges in the graph.
- Do DFS again to create **another DFS forest**.
  - This time, order the nodes in the reverse order of the finishing times that they had from the first DFS run.
- The SCCs are the different trees in the **second DFS forest.**

(Clearly it wasn't obvious since it took so many slides! But hopefully it is less mysterious now.)